UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

USO DE UNA RNN PARA LA GENERACIÓN DE LÍRICAS DE HIP-HOP EN ESPAÑOL

Autor: Oscar Alexander Kirschstein Schafer

Tutor: Miguel Ángel Mora Rincón

MAYO 2021

Resumen

Muchos piensan que actualmente estamos viviendo una cuarta revolución industrial, y uno de los componentes más importantes de ésta es la inteligencia artificial. Independientemente de categorizarse como revolución o no, es imposible negar el impacto de esta herramienta en el mundo actual. El desarrollo de este campo matemático y tecnológico ha permitido el avance en muchos sectores de la industria: economía, recursos humanos, medicina, etc. Entre las mayores fomentadoras de éstos avances se encuentran las redes neuronales artificiales, con toda su familia de evoluciones. Las competencias de esta herramienta innovadora son crear soluciones usando técnicas como la detección, la clasificación o la generación.

Este proyecto se encuentra en el campo de la generación, y su objetivo es llevar a cabo todos los pasos necesarios para la puesta en producción de una red neuronal. En este caso, se trata de una red neuronal recurrente, entrenada para generar líricas de hip-hop en español. Se ha comenzado por realizar un estudio del estado del arte, para seleccionar un modelo adecuado. Posteriormente, se ha desarrollado un conjunto de aplicaciones para recopilar y procesar un conjunto de canciones para entrenar la red, que, una vez entrenada, se ha desplegado en una simple aplicación web con el objetivo de demostrar la capacidad de generación del modelo.

Palabras clave: inteligencia artificial, red neuronal, red neuronal recurrente, generación, hip-hop, líricas

Abstract

Many believe that we are currently experiencing a fourth industrial revolution, and one of the most important components of this is artificial intelligence. Regardless of whether or not it is correct to categorize the current times as a revolution or not, it is impossible to deny the impact of this tool in today's world. The development of this technological and mathematical field has enabled progress in many sectors of industry: economics, human resources, medicine, etc. Among the major drivers of these advances are artificial neural networks, with all their different types. The purpose of this innovative tool is to create solutions using techniques such as detection, classification or generation.
This project is in the field of generation, and its objective is to carry out all the necessary steps for the production of a neural network. In this case, it is a recurrent neural network, trained to generate hip-hop lyrics in Spanish. The first step was to study the state of the art in order to select a suitable model. Subsequently, a series of applications have been developed to collect and process a set of songs to train the network, which, once trained, has been deployed in a simple web application in order to demonstrate the model's generation capacity.

Key words: artificial intelligence, neural network, recurrent neural network, generation, hip-hop, lyrics

Índice general

Índice de figuras

Capítulo 1

Introducción

1.1. Objetivos

El objetivo general de este trabajo ha sido seguir todos los pasos a lo largo de una cadena de producción de un modelo de inteligencia artificial, aplicándolo al desarrollo de un generador de líricas de hip-hop en el idioma español. Además, se puede subdividir el trabajo en cinco secciones principales:

- Llevar a cabo una investigación sobre el estado del arte de la generación de líricas.
- Seleccionar un modelo a partir de la labor de recopilación de información del apartado anterior.
- Crear un conjunto de textos de canciones con el cual alimentar al modelo en la fase de entrenamiento.
- Entrenar el modelo.
- Demostrar el funcionamiento del modelo entrenado. Se hará a través de una aplicación web minimalista.

Todos estos puntos sólo se pueden entender en contexto del objetivo general, que es desarrollar una representación a pequeña escala de la secuencia lógica de una cadena de producción de un proyecto de IA, como se podría encontrar en el ámbito de una empresa, o una investigación científica.

A lo largo de todo el desarrollo se usan una serie de herramientas, comúnmente utilizadas para estas labores, como lo son los cuadernos Jupyter, la librería Keras, Postman o Heroku.

La finalidad de este trabajo no debe entenderse en ningún caso como un intento de lograr un modelo capaz de sustituir el estado del arte contemporáneo.

1.2. Motivación

En esta transformación digital que estamos viviendo desde comienzos de la década pasada, se están logrando avances a una velocidad desorbitada, si se compara con épocas anteriores. Después de que el sector de la inteligencia artificial se estancara en los años 70, hubo un resurgimiento de innovación en los 80, gracias a nuevos algoritmos e inversiones económicas. Debido al incremento exponencial de la capacidad computacional que se ha visto estos últimos años, la IA (Inteligencia Artificial) se ha visto altamente beneficiada, consiguiendo constantes mejoras en las soluciones a problemas [1].

Una de las aplicaciones de esta herramienta es la generación de datos. De generación de texto, como en [2], a generación de cuadros, como en [3], son muchos los proyectos que se han visto en los últimos años. Este es un tema que genera mucha controversia, replanteando cuestiones impermeables al tiempo como la definición de lo que se puede considerar arte o iniciando nuevos debates sociales y políticos, como lo han hecho la aparición de los DeepFakes [4].

En el contexto de este constante desafío a los límites de lo que se creía posible, las motivaciones principales a la hora de realizar este proyecto son dos. Por una parte, ver qué resultados puede conseguir el modelo de IA en la generación de líricas. Y por otra parte, entender cada etapa del proceso que se sigue para poner un modelo en producción, además de conocer distintas herramientas que se pueden utilizar para ello, así como desarrollar otras.

1.3. Organización de la memoria

Esta memoria está dividida en seis capítulos, incluyendo éste, que hace las veces de introducción al proyecto. En el segundo capítulo se expondrá el estado del arte en la generación de líricas, mediante la explicación de una serie de papers. El tercer capítulo contendrá información sobre el diseño y el análisis de las distintas aplicaciones que se han desarrollado para llevar a cabo el proyecto. El cuarto, hablará de la implementación de éstas, dando información sobre las tecnologías utilizadas, y los problemas surgidos. El quinto tratará las pruebas hechas para comprobar el correcto funcionamiento de las aplicaciones y los resultados de la generación de texto. Por último, el sexto capítulo servirá como la conclusión del proyecto, pero también se hablará del posible trabajo futuro que se puede llevar a cabo a raíz de la implementación hecha para el proyecto.

Capítulo 2

Estado del arte

En este capítulo se resumirán cuatro papers sobre la generación de líricas y se presentará el modelo que se ha elegido para llevar a cabo el proyecto. La información obtenida del análisis de los trabajos se ha usado tanto para seleccionar el modelo implementado, como para escoger las herramientas finalmente utilizadas. Cada subsección tendrá el título del nombre del trabajo o modelo descrito en él.

2.1. Automatic Generation of Hip-Hop and Rap Lyrics

[5] es un trabajo de fin de máster en el que se usa aprendizaje profundo para producir líricas de rap. Además de este objetivo primario, tiene un objetivo secundario, el cual consiste en responder algunas cuestiones. Se intenta averiguar si una máquina puede producir líricas de rap, si estas líricas pueden ser consideradas orgánicas y originales, los tipos de rimas que contienen, y cómo se consigue la rima.

Utilizan un corpus en inglés de aproximadamente 11 millones de palabras, contenidas en 1'5 millones de frases. Este corpus se extrajo de la página web [6]. Para llevar a cabo la extracción se utilizó el lenguaje *Python*, haciendo uso de sus módulos *BeautifulSoup* y *Requests* para recolectar las líricas de las canciones.

El corpus se somete a una traducción, para obtener una representación fonética del mismo. Esto se hace en dos pasos. Primero se transforma el texto a una notación fonética denominada *ARPABET*, que identifica los acentos en cada palabra, usando el diccionario CMU. Después, las palabras que no se pudieron traducir con este diccionario se traducen con el conjunto de herramientas de Python llamado G2P.

También se obtuvieron *word embeddings* de las palabras, usando el módulo de Python llamado *Word2Vec*. De esta manera, a parte de la representación fonética, se obtiene también información semántica de cada palabra.

Su modelo esta limitado a generar versos de 13 sílabas, y se fuerza un esquema de rimas AAAA BBBB C DD. Es un modelo denominado como modelo de n-grama invertido.

La generación funciona de la siguiente manera. Primero, se eligen las palabras forzando la rima con el esquema descrito en el párrafo anterior, o sea, las palabras finales de cada línea. Posteriormente es cuando se rellena el resto del verso.

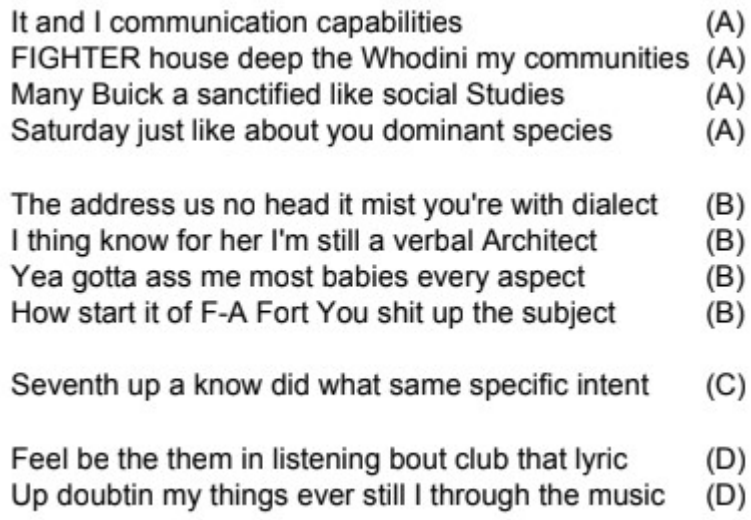

FIGURA 2.1: Ejemplo de texto generado con modelo de n-grama invertido.
Imagen sacada de [5] .

Un ejemplo de líricas generadas con este método se puede observar en la Figura 2.1. Los resultados indican que en este trabajo sí que se consigue que rime con el esquema propuesto, pero semánticamente carece de sentido.

2.2. Lyrics Information Processing: Analysis, Generation, and Applications

En este paper se propone un nuevo campo de estudio llamado *LIP*, que significa *lyrics information processing* [7]. Se argumenta en que es una extensión necesaria del NLP, ya que éste no es suficiente para comprender los aspectos de musicalidad que se encuentran en las líricas. Se introducen tres enfoques principales para este nuevo campo:

1. Análisis de líricas.
2. Generación de líricas y ayuda en la composición.
3. Aplicaciones centradas en líricas.

Dentro del análisis lírico, se introducen ciertas mecánicas:

- **Identificación del esquema de las rimas**
- **Segmentación de líricas**: El esquema de las rimas es algo que se ve analizando línea por línea, pero cada párrafo también es parte de una estructura, como puede ser el estribillo o el coro.
- **Etiquetado**: Darle una etiqueta estructural al tipo de segmento que constituye cada párrafo.
- **Estimación sentimental**: Análisis de los sentimientos que expresa el texto.
- **Modelado de temas**: Análisis de los temas de los que trata el texto.
- **Modelado de historia**: Si se está contando una historia, analizar cómo está estructurada.
- **Análisis de la relación entre texto y audio**.

Dentro de la generación de líricas y ayuda en la composición, se definen los siguientes campos:

- Generación de letras condicionadas por el esquema rítmico.
- Generación de letras condicionadas por la melodía.
- Generación automatizada de líricas estructuradas.
- Escritura fantasma.
- Sistema de apoyo a la escritura con la generación de letras.

Se expone que las tecnologías LIP son útiles para los siguientes tipos de aplicaciones:

- Clasificación de líricas.
- Exploración de líricas.
- Resumen de líricas.

2.3. DopeLearning: A Computational Approach to Rap Lyrics Generation

En este paper se defiende que escribir hip-hop/rap requiere de habilidades líricas y creatividad. Consecuentemente, se presenta un modelo que pretende implementar ambas. El generador ha sido implementado en una aplicación web llamada *DeepBeat* [8].

La generación de letras se enfoca como un problema de IR (Information Retrieval). Parten de una base de datos de aproximadamente medio millón de líneas reales, de 104 artistas diferentes. Primero, hacen una selección de un número selecto de líneas, las cuales hacen el papel de consulta. Ésta, se le pasa al modelo, que devolverá las líneas más relevantes a las de la consulta, sacándolas de la base de datos. Es decir, la letra generada se compone de diversas líneas de canciones ya existentes.

Su método consiste de dos bloques. Por una parte, un modelo lingüístico neural, que se encargará de detectar la similitud semántica, y por otro, el modelo RankSVM (Ranking Support Vector Machine) usado para combinar las distintas características de cada verso y calcular su relevancia. Cabe destacar que se filtran ciertos versos para evitar que dos líneas consecutivas pertenezcan a la misma canción y asegurar que rimen.

2.4. GhostWriter: Using an LSTM for Automatic Rap Lyric Generation

En este paper se define un modelo capaz de emular la escritura fantasma, es decir, la generación de letras que siguen el estilo de un artista, pero que difieren de las líricas de sus canciones ya existentes [9]. Además del estilo del artista, el sistema debe ser capaz de modelar el lenguaje humano de manera genérica.

Además de letras, se genera también una estructura del texto. Para ello se introducen tokens especiales en el texto. Se puede apreciar una muestra de unas líricas generadas con este modelo en la Figura 2.2.

El conjunto de datos se compone de 14 artistas cuyas letras se han recolectado de la página web [6]. El tipo de modelo seleccionado es el LSTM (Long Short-Term Memory), un tipo de modelo predecesor al que se ha utilizado en este TFG (Trabajo de Fin de Grado). Este modelo demuestra ser efectivo para la tarea propuesta y, además, mejor que el modelo de n-grama.

2.5. Bi-LSTM

En esta sección se presentará el modelo usado para los propósitos de este proyecto. La razón detrás de la decisión de por qué utilizar éste modelo, es su exitoso uso en varios artículos académicos relacionados con la generación de texto, como por ejemplo en [9].

Este modelo es el Bi-LSTM, o LSTM bidireccional. Las LSTM son un tipo de RNN (Recurrent Neural Network). Sus siglas significan larga memoria de corto plazo. Al contrario de la clasificación estática que pueden proveer tipos de modelos como las ANN

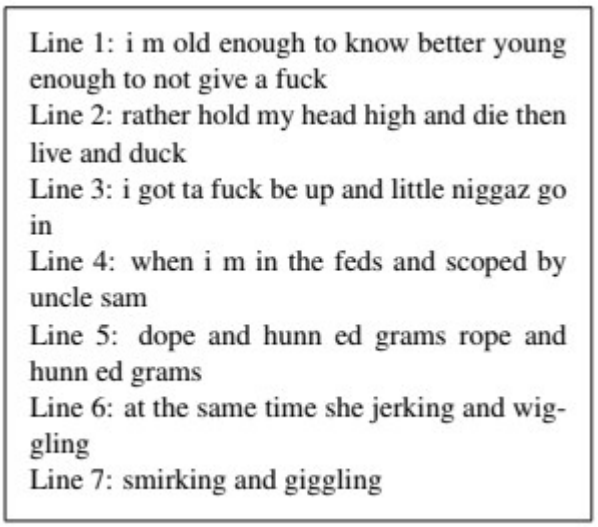

FIGURA 2.2: Ejemplo de texto generado con el modelo LSTM Ghostwriter.
Imagen sacada de [9] .

(Artificial Neural Network), los modelos RNN nos ofrecen una clasificación dinámica, ya que una clasificación puede depender de qué se haya clasificado anteriormente.

Pero las RNN tenían un problema, y es que el valor del que dependían para actualizar sus memorias, es decir, el valor que las hace aprender, se volvía o demasiado grande o demasiado pequeño tras aproximadamente diez iteraciones de clasificaciones. Esto dificulta y ralentiza tremendamente el aprendizaje. La solución a esto son las LSTM, que evitan que este valor explotase o desvaneciese.

Sin embargo, las LSTM también tienen un problema. Es uno típico en las RNN, y es que, al aprender, sólo analizan las palabras pasadas al hacer una nueva predicción. Las Bi-LSTM sin embargo, superan esta limitación en el aprendizaje al entrenar simultáneamente dos modelos LSTM. Uno se encarga de analizar las palabras pasadas, y otro se encarga de las palabras futuras. Ambos están conectados a la misma capa de salida, pudiendo aprovechar así toda la información obtenida [10].

Capítulo 3

Análisis y diseño

En este capítulo se comentan las distintas decisiones de análisis y de diseño que se han tomado para desarrollar el proyecto. También hay un apartado para cada una de las distintas aplicaciones de las que se compone el proyecto. Para cada aplicación, hay secciones dedicadas a los requisitos funcionales y no funcionales respectivamente. También se incluyen algunos diagramas para facilitar la comprensión del diseño de las aplicaciones.

La funcionalidad del conjunto de aplicaciones y su interconexión se describen de manera genérica en el diagrama de actividades de la Figura 3.1. Hay cinco aplicaciones distinguidas pero conectadas entre sí. :

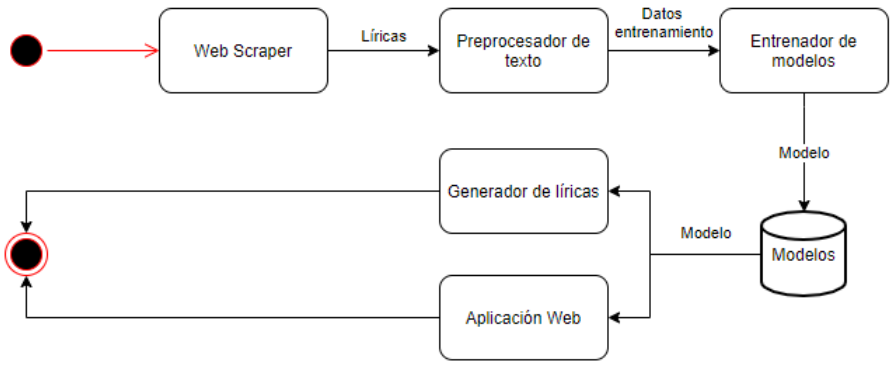

FIGURA 3.1: Diagrama de actividades general

1. **Web Scraper**: Se encarga de recolectar las líricas de ciertos artistas preseleccionados.

2. **Preprocesador de texto**: Se encarga de transformar las líricas en un conjunto de metadatos y datos legibles por el modelo.

3. **Entrenador de modelos**: Permite entrenar distintos modelos de generación de líricas.

4. **Generador de líricas**: Permite una generación parametrizada de líricas con los modelos entrenados.

5. **Aplicación web**: Hace una demostración de la generación de líricas, como la aplicación anterior, pero esta vez a través de una interfaz web.

3.1. Web Scraper

Esta sección habla de las decisiones de análisis tomadas para la aplicación de *web scraping*.

El objetivo general del *Web Scraper* es contactar con la API (Application Programming Interface) de *Genius* ([11]) y descargar todas las líricas que pertenezcan a unos artistas seleccionados manualmente. Para ello, se hace uso de un archivo CSV (Comma Separated Values) que contiene el nombre de cada artista junto a su identificador en la base de datos de *Genius*. También le muestra al usuario información relevante durante el proceso de descarga.

3.1.1. Requisitos funcionales

- **WS-RF1.** El usuario deberá poder descargar las líricas de unos artistas preseleccionados por él.
- **WS-RF2.** Deberá poder realizar la descarga en varias iteraciones si así lo desea.
- **WS-RF3.** Deberá mostrar las letras descargadas para cada artista sobre el total de letras del artista.
- **WS-RF4.** Deberá mostrar los artistas para los que ya fueron descargadas las letras anteriormente.

3.1.2. Requisitos no funcionales

- **WS-RNF1.** En caso de interrupción de la ejecución del programa, las letras ya descargadas deben quedar guardadas.
- **WS-RNF2.** Se deberá mostrar al usuario el proceso de descarga de las letras.
- **WS-RNF3.** La aplicación debe estar implementada para línea de comandos.

3.2. Preprocesador de texto

Esta sección habla de las decisiones de diseño de software tomadas para la aplicación de preprocesamiento de texto.

Esta aplicación tiene la función de cargar las letras descargadas por el *Web Scraper* y procesarlas, normalizándolas, y extrayendo los metadatos necesarios para entrenar un modelo.

3.2.1. Requisitos funcionales

- **PT-RF1.** El usuario deberá poder elegir los artistas de los que generar el corpus.
  - **PT-RF1.1** Podrá escoger cualquier cantidad de artistas de entre los que hay disponibles.
  - **PT-RF1.2** Podrá elegir abortar el procedimiento durante esta fase de selección.
- **PT-RF2.** Al final del procesamiento se deberán guardar todos los datos necesarios para iniciar el entrenamiento de un modelo.
- **PT-RF3.** Mostrar el número de canciones procesadas.
- **PT-RF4.** Mostrar los artistas que se eligieron para el procesamiento.
- **PT-RF5.** Mostrar las palabras únicas que hay antes de eliminar las que no se consideran válidas.
- **PT-RF6.** Mostrar el mínimo número de veces que debe aparecer una palabra para que se considere válida.
- **PT-RF7.** Mostrar el número de palabras únicas que hay después de eliminar las no válidas.
- **PT-RF8.** Mostrar el número de secuencias de entrenamiento que se ignoran debido a la eliminación de palabras.
- **PT-RF9.** Mostrar el número de secuencias de entrenamiento finales.

3.2.2. Requisitos no funcionales

- **PT-RNF1.** El programa deberá indicar cuando se ha completado el preprocesamiento.
- **PT-RNF2.** La aplicación deberá mostrar al usuario metainformación relevante al preprocesamiento.
- **PT-RNF3.** La aplicación debe estar implementada para línea de comandos.

3.3. Entrenador de modelos

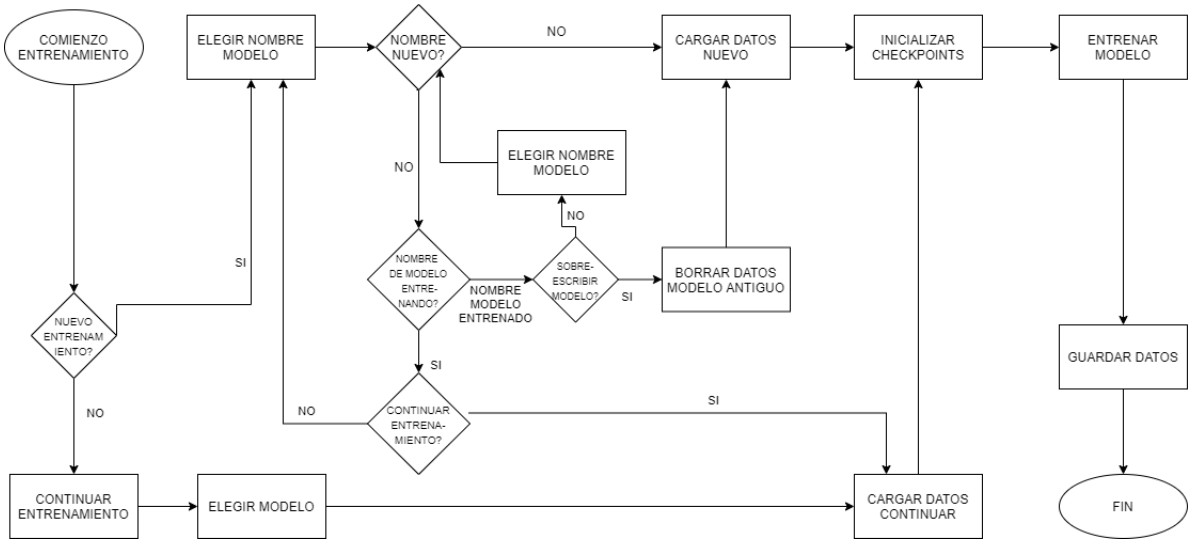

FIGURA 3.2: Diagrama de flujo de datos del entrenador de modelos.

En esta sección se señalarán las decisiones de diseño y análisis tomadas para la aplicación de entrenamiento de modelos. A parte de la lista de requisitos funcionales y no funcionales, se incluye un DFD (Diagrama de Flujo de Datos) que muestra el funcionamiento del programa.

La aplicación da la posibilidad al usuario de entrenar un modelo nuevo usando los últimos datos que ha generado el preprocesador, o continuar entrenando un modelo cuyo entrenamiento ha sido interrumpido. En este último caso, los datos que se usan son los últimos que generó el preprocesador en el momento de empezar a entrenar el modelo por primera vez. Esto es posible ya que éstos se guardan junto a los puntos de guardado del modelo.

En la Figura 3.2 se puede observar el diagrama de flujo de datos que describe el funcionamiento de esta aplicación.

3.3.1. Requisitos funcionales

- **EM-RF1.** El usuario deberá poder entrenar un modelo nuevo. Para ello elegirá un nombre para el modelo.
    - **EM-RF1.1.** Si el nombre del modelo elegido corresponde a un modelo en proceso de entrenamiento, se debe dar la opción de continuar entrenándolo.
    - **EM-RF1.2.** Si el nombre del modelo elegido corresponde a un modelo ya entrenado, se debe dar la opción de sobreescribirlo.
    - **EM-RF1.3.** Si el nombre del modelo elegido es único, se comenzará el entrenamiento.

- **EM-RF2.** El usuario deberá poder seguir con el entrenamiento de un modelo. Para ello elegirá uno de los modelos cuyo entrenamiento ya ha empezado.
- **EM-RF3.** Al comenzar el entrenamiento se guardará el modelo inicializado.
- **EM-RF4.** Durante el entrenamiento, tras finalizar cada época, quedará guardada la mejor versión del modelo.
- **EM-RF5.** Al final del procesamiento se deberán guardar todos los datos necesarios para realizar una generación de líricas.
- **EM-RF6.** Se mostrará el tamaño del conjunto de entrenamiento.
- **EM-RF7.** Se mostrará el tamaño del conjunto de prueba.
- **EM-RF8.** Se mostrará el progreso del entrenamiento.
- **EM-RF9.** Se debe poder salir del programa en cualquier punto de éste, escribiendo la secuencia "*ESC*" en la línea de comandos cuando esté activa.

3.3.2. Requisitos no funcionales

- **EM-RNF1.** El programa se podrá interrumpir en cualquier punto del entrenamiento, quedando persistente en la memoria el estado del entrenamiento hasta la última mejor época completada.
- **EM-RNF2.** El programa mostrará información relevante al entrenamiento.
- **EM-RNF3.** La aplicación debe estar implementada para línea de comandos.

3.4. Generador de líricas

Esta sección tratará el análisis y diseño planteado para la aplicación del generador de líricas.

Esta aplicación completa el ciclo de generación de letras mostrando una letra creada por uno de los modelos completamente entrenados, el cual se selecciona interactivamente. Se hace a través de una interfaz de línea de comandos.

3.4.1. Requisitos funcionales

- **GL-RF1.** El usuario debe elegir el modelo con el que generar líricas.

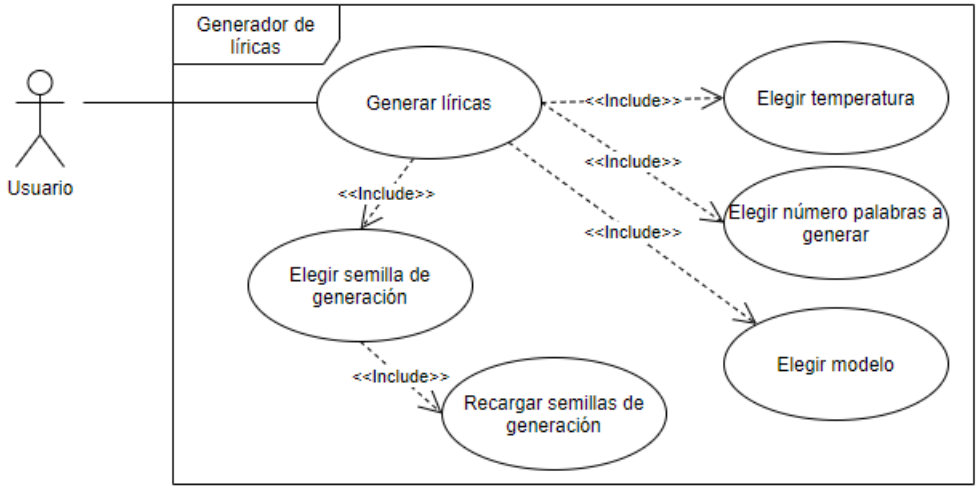

FIGURA 3.3: Diagrama de casos de uso del generador de líricas por línea de comandos.

- **GL-RF2.** Se deberá poder elegir una opción como frase-semilla de generación de entre las siguientes:
  - **GL-RF2.1.** Una semilla específica eligiendo de entre las frases mostradas.
  - **GL-RF2.2.** Una semilla escogida de modo aleatorio del conjunto total de semillas.
- **GL-RF3.** Se debe poder recargar las semillas mostradas para selección.
- **GL-RF4.** El usuario debe elegir el número de tokens a generar, siendo este número un valor entre 20 y 500.
- **GL-RF5.** El usuario debe elegir el grado de diversidad de la generación, siendo este un número real en el rango (0,1].
- **GL-RF6.** Al final de la generación se mostrará:
  - **GL-RF6.1.** La letra generada.
  - **GL-RF6.2.** La diversidad con la que se ha generado la letra.
  - **GL-RF6.3.** Los nombres de los artistas cuyas letras se han usado en el entrenamiento del modelo usado para la generación.
  - **GL-RF6.4.** La frase que ha servido como semilla de la generación.
- **GL-RF7.** Se debe poder salir del programa en cualquier punto de éste, escribiendo la secuencia *"ESC"* en la línea de comandos cuando esté activa.

3.4.2. Requisitos no funcionales

- **GL-RNF1.** A la hora de seleccionar los modelos, éstos se deben mostrar junto a los nombres de los artistas con cuyas letras se han entrenado.

- **GL-RNF2.** Se debe mostrar información sobre la generación.
- **GL-RNF3.** Se debe avisar al usuario cuando éste ha introducido valores erróneos para la diversidad o para el número de tokens a generar.
- **GL-RNF4.** La aplicación debe estar implementada para línea de comandos.

En la Figura 3.3 se puede observar el diagrama de casos de uso del generador de líricas.

3.5. Aplicación web

Esta sección hablará de las decisiones de diseño y análisis tomadas para la aplicación web.

La aplicación es muy similar al generador de texto. La diferencia principal es que es una interfaz web la cual permite la generación de texto parametrizada. Otro aspecto en el que difieren estas aplicaciones es que en esta no se permite al usuario el elegir el modelo con el que hacer la generación, ya que ya está predefinido.

3.5.1. Requisitos funcionales

- **AW-RF1.** Deberá poder elegir una opción como frase-semilla de generación de entre las siguientes:
  - **AW-RF1.1.** Una semilla específica eligiendo de entre las frases mostradas.
  - **AW-RF1.2.** Una semilla escogida de modo aleatorio del conjunto total de semillas.
- **AW-RF3.** Se debe poder recargar las semillas mostradas para selección.
- **AW-RF4.** El usuario puede elegir el número de tokens a generar, siendo este número positivo menor que 501.
- **AW-RF5.** El usuario puede elegir el grado de diversidad de la generación, siendo este un número real en el rango (0,1].
- **AW-RF6.** El modelo usado para la generación debe estar predefinido como variable estática antes del despliegue de la aplicación.
- **AW-RF7.** Al final de la generación, se mostrará:
  - **AW-RF7.1.** La frase que ha servido como semilla de la generación.
  - **AW-RF7.2.** La diversidad con la que se ha generado la letra.

- **AW-RF7.3.** El número de tokens que se han generado.
- **AW-RF7.4.** Los nombres de los artistas cuyas letras se han usado en el entrenamiento del modelo usado para la generación.

3.5.2. Requisitos no funcionales

- **AW-RNF1.** Se debe mostrar unas instrucciones de uso del servicio de generación.
- **AW-RNF2.** Una vez la letra ha sido generada, se debe mostrar al usuario acompañada de la información relevante a ésta.
- **AW-RNF3.** Se debe avisar al usuario cuando éste ha introducido valores erróneos en el formulario.
- **AW-RNF4.** Se debe resaltar la semilla seleccionada con un color distinto al de fondo.

Se puede observar el diagrama de casos de uso de la aplicación web en la Figura 3.4. Ésta es muy similar a la Figura 3.3, ya que tienen la misma finalidad, que es la generación de líricas a partir de un modelo. La diferencia es que en este caso, la elección de diversidad y número de palabras no es obligatoria, y además, no hay que elegir modelo ya que sólo hay un modelo desplegado, previamente seleccionado de manera estática durante la programación de la aplicación.

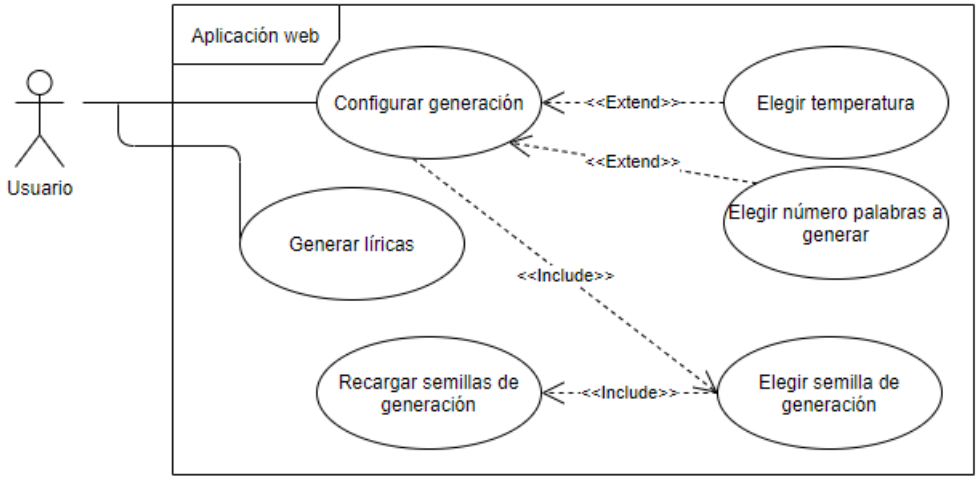

FIGURA 3.4: Diagrama de casos de uso de la aplicación web generador de líricas.

Capítulo 4

Desarrollo

Este capítulo sirve para comentar los pasos y decisiones que se han tomado a lo largo del desarrollo de este proyecto. El proyecto se ha desarrollado en dos subproyectos. El primero, dedicado a los cuatro scripts de web scraping, preprocesamiento, entrenamiento de modelos y generación de texto. El segundo, dedicado a la aplicación web. En la primera sección se describe el entorno usado durante el desarrollo, y después, cada capítulo trata de un apartado del proyecto distinto.

Como complemento a esta sección, en el Anexo C se puede encontrar un manual de usuario detallado, además de capturas de pantalla informativas, para cada una de las aplicaciones de las que se compone el proyecto.

4.1. Entorno

En esta sección se comentan todos los elementos de entorno destacables de los que se ha hecho uso en este proyecto.

4.1.1. Plataformas

Esta sección describe las plataformas que se usan en este proyecto.

Windows 10

Es el sistema operativo en el que se ha desarrollado el proyecto en su totalidad. Se ha decidido usar este SO (Sistema Operativo) debido a su facilidad de uso y su familiaridad. Es software propietario desarrollado por Microsoft. [12]

Visual Studio Code

Visual Studio Code es un editor de código lanzado por Microsoft en 2015. Se ha elegido como entorno de programación por varias razones. Es compatible con Python y con Jupyter Notebooks, dos herramientas utilizadas en el proyecto. Además es ligero y tiene algunas extensiones muy convenientes como la integración con git o la recomendación de texto que han resultado muy útiles a la hora de trabajar eficientemente. [13]

Heroku

Heroku es una plataforma en la nube que implementa el modelo de negocio PaaS (Platform as a Service), o plataforma como servicio. Permite gestionar, desplegar, servir y monitorizar aplicaciones, ofreciendo ventajas como una fácil escalabilidad e integración con la herramienta git. Se ha seleccionado para este proyecto, para desplegar la aplicación web. La razón es que tiene una modalidad de uso gratuito para aplicaciones que ocupen menos de 500MB de espacio en disco. [14]

4.1.2. Herramientas

En esta sección se explican todas las herramientas principales de las que se hace uso en este proyecto.

Git

Git es un sistema de control de versiones desarrollado por Linus Torvalds en 2005 y utilizado en el desarrollo del kernel de Linux. Normalmente se utiliza para coordinar proyectos de programación. Tiene muchas funcionalidades que sirven para llevar un proyecto a través de múltiples fases del desarrollo, pudiendo incluso volver a examinar versiones en cualquier punto de éste. Se ha elegido para el proyecto debido al potente abanico de funciones que ofrece, como el rastreador de cambios, y a la familiaridad con él. En concreto se ha optado por la implementación que ofrece la plataforma de desarrollo GitHub, que es utilizada por más de 65 millones de desarrolladores en todo el mundo. [15]

Python

Python es el lenguaje de programación orientado a objetos que se ha usado para implementar este proyecto. Es administrado por la Python Software Foundation. Ofrece ventajas como una sintaxis fácilmente entendible y una gran librería estándar. Tiene un modo interactivo, que puede ser usado para hacer pruebas rápidas y a las que otras herramientas como Jupyter Notebook sacan partido. Algunas características útiles son la posibilidad de usar generadores o la gestión de memoria automática. Para la implementación de este proyecto usamos su versión 3.8.2 de 64 bits. [16]

Jupyter Notebook

Es una aplicación web que forma parte del Proyecto Jupyter, un proyecto, de código abierto nacido en 2014, con la intención de ofrecer una herramienta para el trabajo científico y la ciencia de datos. Jupyter notebook es una herramienta comúnmente utilizada en el ámbito de la investigación, con la que se puede programar en Python. Permite ejecutar celdas de código por separado, permitiendo una fácil modularización del código. Además de esto, es una buena opción para hacer pruebas, antes de implementar el código en *scripts*. En este caso, se ha utilizado para hacer una implementación completa del pipeline de producción (menos la aplicación web), antes de su división en cuatro aplicaciones distintas. [17]

Flask

Flask es un framework implementado en Python diseñado para la creación de aplicaciones web de manera rápida y eficiente, utilizando un número muy pequeño de líneas. Tiene funcionalidades muy útiles como el uso de plantillas dinámicas de HTML o un intuitivo motor de enrutamiento. Opera bajo la licencia BSD y fue desarrollado por Armin Ronacher. [18]

Postman

Postman es una plataforma de colaboración gratuita para el desarrollo de APIs. En este caso, se ha utilizado para comprobar la funcionalidad de la API REST de la Genius API: la fuente de nuestras letras. Se ha elegido ya que ofrece una interfaz intuitiva para el envío de peticiones HTTP (HyperText Transfer Protocol) con parámetros personalizados y la visualización de la respuesta. Además, su plan gratuito se adaptaba a las necesidades de nuestro proyecto. [19]

4.1.3. Librerías

En esta sección se describirán brevemente las librerías de Python destacables utilizadas en el proyecto.

Pandas

Pandas es un paquete de Python que ofrece un gran arsenal de herramientas para un manejo de datos de manera fácil e intuitiva. Es muy útil para el análisis de datos. Su principal estructura, el *DataFrame*, tiene muchos paralelismos con una tabla en una base de datos relacional. Entre las funcionalidades que destacan están las funciones de agrupación, la habilidad de aplicar funciones de manera legible y eficiente a tablas de datos o las diferentes funcionalidades de indexado. Muchos de los algoritmos de bajo nivel han sido escritos en Cython, que permite combinar Python y C consiguiendo unos

tiempos de ejecución mucho más optimizados. Es una librería de código abierto, con una licencia BSD. [20]

Requests

Requests es una librería de alto nivel para el manejo de peticiones HTTP con Python. Permite el envío de peticiones HTTP de manera intuitiva así como el recibimiento de las respuestas, a través de este lenguaje. Es muy útil para la comunicación con APIs web. Esta librería fue implementada por Kenneth Reitz. En este proyecto se ha usado en consonancia con la herramienta Postman. [21]

Bs4

BS4 o Beautifoul Soup 4, es la última versión de las librerías Beautiful Soup, y la única con soporte activo. Esta librería hace las veces de extractora de información de archivos HTML o XML. También es capaz de navegar y modificar el árbol de elementos del que se componen este tipo de archivos. Por defecto, utiliza el parseador HTML por defecto incluido en la librería estándar de Python, pero se puede configurar para su uso con analizadores sintácticos desarrollados por terceros. [22]

NumPy

NumPy es un proyecto de código abierto que sirve para llevar a cabo todo tipo de operaciones de cálculo con números. En 2005 fue creado y opera bajo una licencia BSD modificada. Se desarrolla de manera transparente en la plataforma GitHub. Destaca por la sencillez con la que permite operar con matrices, por ejemplo. Esto es una gran ventaja para los proyectos de inteligencia artificial, y sobre todo a la hora de desarrollar algoritmos a bajo nivel. Similar a Pandas, el núcleo de numpy se compone de código C altamente optimizado. [23]

Re

Es una librería incluida en la librería estándar de Python que permite trabajar con expresiones regulares. Es un paquete muy útil para proyectos de NLP como éste, a la hora de llevar a cabo el preprocesado de texto, por ejemplo, permitiendo una fácil maleabilidad del corpus lingüístico utilizado para entrenar un modelo. [24]

Pickle

Pickle es un paquete de Python que implementa protocolos para la serialización y deserialización de objetos. Por ello, es muy útil a la hora de guardar y cargar datos del disco duro. [25]

Glob

Glob es una librera de Python que une la capacidad de síntesis que ofrecen las expresiones regulares y la navegación por una jerarquía de nombres de carpetas y archivos. Es útil a la hora de buscar archivos que sigan un patrón. Se puede utilizar en el contexto de un proyecto de IA p.e. para encontrar el archivo del punto de guardado más reciente de un modelo. [26]

Keras

Keras es una interfaz de programación de Python especializada en aprendizaje profundo. En este proyecto, se usa la implementación que corre sobre un backend de Tensorflow 2, que es una librería que sirve como plataforma de *machine learning* de código abierto. Keras funciona como una interfaz de alto nivel de Tensorflow 2, ofreciendo la posibilidad de implementar casos comunes de piezas clave de programas de IA con pocas líneas de código, manteniendo una alta legibilidad i.e. funciones objetivo, funciones de activación, capas u optimizadores. El desarrollador de Keras es François Chollet, un ingeniero de Google. La integración con Tensorflow se produjo de manera oficial en 2017, y apartir de la versión 2.4, Tensorflow se convierte en el único back-end al que da soporte Keras. [27]

4.2. Artistas e identificadores

Ya habiendo elegido el modelo, la primera labor fue reunir un conjunto de datos sobre el que trabajar. Para ello se buscaron varias páginas de líricas. Se encontraron muchas más opciones para líricas en inglés que de español, pero finalmente, se decidió usar la página de Genius para conseguir las líricas, ya que ofrecía una API REST con la cual poder acceder a la información de sus bases de datos. Para poder utilizarla, hay que registrarse para obtener el token de autenticación que se manda junto a cada petición.

Se encontró una librería de Python llamada *lyricsgenius* ([28]), que permite realizar la acción de scraping de manera intuitiva a alto nivel. Aún así, se decidió no hacer uso de ésta por que algunos de los artistas seleccionados daban problemas de ambigüedad al buscarlos con el motor de búsqueda de la librería, ya que había varios artistas con nombres similares. Por ello, finalmente se decidió programar un rastreador web por cuenta propia.

Se seleccionaron 21 artistas manualmente, cuyas canciones se podrían usar para entrenar el modelo final. Estos artistas son: C Tangana, Cecilio G, Chaman, Charlie, Delaossa, Denom, Dudu, Easy-S, El Jincho, Hard GZ, Israel B, Ivan Cano, Kase O, Lopes, Natos & Waor, Nikone, Oktoba, Recycled J, Tote King, Walls y Yung Beef.

Un atributo importante para poder recuperar las canciones, y con ello, las líricas de cada artista fue conocer el id de los respectivos cantantes. Para ello, se utilizó la herramienta Postman. Se mandó la siguiente petición HTTP a la API de Genius , donde se sustituye $<$ *nombre_artista* $>$ con el nombre del artista solicitado:

- api.genius.com/search?q=$<$ *nombre_artista* $>$

Esta petición devuelve un resultado de búsqueda dentro de la base de datos de Genius, en formato JSON, que contiene varios objetos, la mayoría de ellos canciones. Dentro de las canciones, se buscó manualmente aquellas en las que el artista buscado era el autor principal, ya que en esas, también se encontraba el identificador del cantante. Junto a éste, se encontraba un link que llevaba a una página HTML, en la que se podía comprobar fácilmente si el artista seleccionado era el correcto. Esto explica por qué se seleccionó artistas familiares en un principio.

Una vez realizado este procedimiento manualmente para cada artista de la lista, se hizo un archivo CSV llamado *Artistas_IDs.csv* con dos columnas, en las que están relacionados el nombre del artista con su identificador. Con esto, se tiene suficiente información para continuar haciendo el *Web Scraper*.

4.3. Modelo

Otro aspecto a tener en cuenta que surgió durante el desarrollo del cuaderno fue la arquitectura del modelo que se implementaría. Como se comentó anteriormente, se eligió un modelo Bi-LSTM, debido a los buenos resultados que produce [9]. Finalmente, se eligió inspirarse en la implementación de [29].

A parte de la arquitectura del modelo, había otras decisiones que hacer. Una de ellas fue enfocar el modelo para la generación palabra a palabra. En [8], por ejemplo, se emplea una generación frase a frase, pero esta requiere de un muy amplio repertorio de canciones, del que no se disponía en este caso. Otro enfoque era el de generar letra por letra, pero como se concluye en [30], esto es computacionalmente muy costoso, lo cual incrementaría aún más los ya altos tiempos de entrenamiento, por lo que se rechazó también esta opción.

Se hicieron varias iteraciones de modelos hasta llegar a la versión final. A lo largo de estas iteraciones una de las labores fue ajustar los parámetros del código de manera manual, no sistemática, hasta obtener resultados aceptables. Estos parámetros están documentados en el Anexo D. Otro cambio que se introdujo en una de las iteraciones fue el de entrenar usando la técnica de *word embeddings*, aprovechando que la librería Keras ofrece una capa llamada *Embedding* con esta funcionalidad. Con esto lo que se consigue

es que el modelo aprenda una representación vectorial de cada palabra, con la finalidad de que palabras que tengan una relación semántica más estrecha, estén más cerca en el espacio euclídeo. Esto mejoró los resultados de la predicción notablemente, pasando de una accuracy de 61 % a una de más del 70 %.

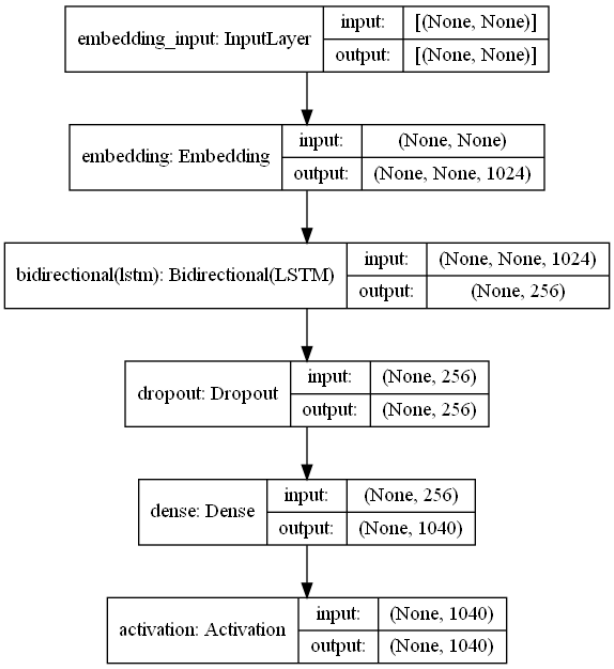

FIGURA 4.1: Arquitectura del modelo de generación de texto.

4.4. Formatos

Todas las aplicaciones producen datos para el uso de otra aplicación, menos la interfaz web y el generador de texto por línea de comandos. El formato elegido para que las distintas aplicaciones carguen y guarden datos, que no son parte de la estructura de un modelo, es el protocolo 5 de pickle, definido en [31]. Los puntos de guardado de los modelos, y los modelos finales se guardan en formato *HDF5* (descrito en [32]), abreviado con la extensión *h5*. Otra excepción es el archivo de artistas e identificadores, que está en formato CSV.

4.4.1. Muestreo

Una vez entrenado un modelo en el cuaderno de prueba, se ha llegado al punto de poder generar texto. Para ello, se debe elegir una frase como semilla. Esta será de la misma longitud que las secuencias de entrenamiento, para ser compatible con el modelo. Cabe destacar que se podría utilizar cualquier combinación de letras de esa longitud como semilla, mientras que las palabras también estén presentes en el conjunto de datos de entrenamiento. En cambio, en este proyecto sólo utilizan como posibles semillas frases extraídas de las secuencias de entrenamiento, con la condición de que que comiencen

al principio de un verso, para servir como inicio del texto generado. Esto ha sido por inspiración de [9], ya que proporciona una buena base con sentido semántico.

Se decidió realizar esto utilizando el muestreo temperamental, una técnica comúnmente usada en la generación de texto [33]. Esta técnica utiliza un parámetro para construir el modelo probabilístico a partir de la salida del generador de texto. Este parámetro, llamado temperatura, afecta a las probabilidades finales de predicción. Cuanto más cercano sea a cero este valor, más confiará el modelo en los valores altos que asigne a cada palabra. Esto tiene el efecto de que, si la temperatura elegida para el muestreo es baja, se generarán letras muy parecidas a las originales, es decir, a aquellas con las que se entrenó el modelo. Pero si se usa una temperatura demasiado alta, es más probable de que el texto generado carezca de sentido semántico alguno.

4.5. Web Scraper

Una vez conseguido el archivo que relaciona a cada artista con el número que lo identifica en la API de Genius, se procedió a hacer el Web Scraper, implementado en un script. Este programa tiene permiso para lanzar peticiones HTTP a la API ya que usa un token de autenticación obtenido manualmente a través de la web de la API. Cabe destacar que esta API no tiene ningún límite de peticiones documentado.

El objetivo de este programa es construir una tabla para cada artista, que se guardará en disco. El archivo tendrá será nombrado igual que el artista cuyas canciones contiene. Cada una de esas tablas tiene dos columnas con la siguiente información:

- Identificador de la canción.
- Letra de la canción.

Antes de generar estas tablas, primero se construye una tabla auxiliar, con los metadatos de las canciones. Ésta contiene cuatro columnas, que contienen la siguiente información:

- Identificador de la canción.
- URL (Uniform Resource Locator) de las líricas de la canción.
- Título de la canción.
- Nombre del artista al que pertenece la canción.

4.5.1. Tabla auxiliar

La tabla auxiliar se construye iterando sobre cada artista del archivo que los relaciona con su identificador. Con el identificador, se solicita a la API la primera página de las canciones en las que aparece el artista, con la siguiente petición, donde <*id_artista*> es el valor del ID de cada artista. Además, se pasa como parámetro el valor 1 para <*page*>.

- https://api.genius.com/artists/<*id_artista*>/songs?page=<*page*>

Una vez obtenida esta primera página en formato JSON, se itera sobre cada canción anotando la información pertinente, siendo muy importante la URL donde encontrar las líricas. Otra parte de la respuesta es el elemento que indica el número de la siguiente página de canciones. Si este valor es igual a *None*, entonces significa que el artista ya no tiene más canciones en la base de datos , y por lo tanto se pasa a procesar al siguiente. Si es un número real, significa que tiene más páginas de canciones, y por lo tanto se vuelve a hacer una petición a la API modificando el parámetro <*page*>.

Cuando ya se han procesado todos los artistas, se guarda la tabla como un *DataFrame*. Esto se hace ya que este procedimiento es largo, y primero se comprueba si ya existe este archivo, para poder obviar la creación de la tabla auxiliar, e ir directamente al siguiente paso.

4.5.2. Tablas finales

Para construir las tablas finales, con las letras de las canciones de cada artista ya descargadas, se hace uso de la tabla auxiliar. Para cada artista, se itera sobre las URLs de sus canciones. Con cada una, se obtiene un fichero HTML, que se parsea usando la librería BS4. Una vez extraídas la letras, se guardan junto a sus identificadores en la tabla final para ese artista.

Durante el parseo, surgió un problema, y es que se observó que sólo se descargaba una parte de las líricas. Después de investigar el asunto, se descubrió que había dos estructuras de HTML distintas, y el procedimiento de parseo hasta entonces sólo encontraba las líricas en una de ellas. Sabiendo esto, se modificó apropiadamente el código de parseo, logrando que finalmente se descargaran todas las canciones para todos los artistas.

4.6. Preprocesador de texto

Una vez descargadas las letras de los artistas, se procedió a hacer el script para procesar éstas y generar datos con los que se pudiera entrenar a la IA. A parte del procesamiento de las letras en sí, se generan metadatos necesarios para el entrenamiento.

4.6.1. Procesamiento de líricas

El primer paso del script es realizar el procesamiento de las líricas, normalizándolas. Este proceso se lleva a cabo para eliminar posibles secuencias de caracteres o números que no aporten información semántica al modelo a entrenar.

Se comienza mostrando al usuario la selección de artistas que tiene disponible para crear el conjunto de entrenamiento, instándole a elegir cualquier combinación de ellos para proceder a procesar los textos de sus canciones. También se le muestra de qué artistas se compone el conjunto de datos actualmente existente. Sólo puede haber un conjunto de líricas procesadas a la vez. Una vez elegidos los artistas, se comienza a procesar, pasándole a cada canción una secuencia de funciones, que implementan las técnicas de procesamiento del apartado a continuación.

Técnicas de procesado

- **Eliminación de guiones**: En algunas canciones se pueden encontrar guiones que simbolizan una conversación. Estos los eliminamos.
- **Eliminación de texto previo al título**: Algunas canciones contienen texto informativo previo al título, que no pertenece a las líricas, por tanto lo quitamos.
- **Eliminación de comentarios**: El texto contiene etiquetas que indican qué funcionalidad cumple cierto párrafo en la canción, i.e. estribillo. Si la canción está interpretada por múltiples artistas, el se puede encontrar etiquetas indicando quién canta cada parte. Esta información se elimina.
- **Eliminación salto de línea doble**: Se elimina el salto de línea doble que hay en algunas canciones ya que no se distinguirá entre párrafos en el modelo.
- **Eliminación signos de puntuación**: Se eliminan todos los signos de puntuación usando una expresión regular.
- **Corregir saltos de línea**: Algunos documentos se descargan de tal manera que no hay saltos de líneas salvo en los cambios de estrofa. Hay una manera de detectar donde meter el salto de línea, ya que el comienzo de línea siempre es con una mayúscula, y el final siempre con minúscula. Por tanto los saltos de línea hay que introducirlos entre dos caracteres cuando se trate de una letra minúscula seguida de una mayúscula.
- **Cambiar todo a minúsculas**: Esto se hace con el objetivo de que dos palabras con el mismo se significado no se consideren distintas por llevar mayúsculas o no llevarlas.
- **Permitir tokenización del salto de línea**: El salto de línea en la mayoría de los casos dentro del texto es adyacente a dos caracteres que no son espacios en blanco. Esto, como se verá en la tokenización provocaría un error, ya que se las palabras

inmediatamente adyacentes y el salto de línea se considerarían como un sólo token. Por ello se introduce un espacio en blanco inmediatamente antes y después de este.

4.6.2. Tokenización

Una vez procesado el texto, se procede a tokenizarlo. Esto significa dividir el texto en unidades que van a poder ser generadas por el modelo de IA. Se aceptan como tokens todas las cadenas de caracteres menos las cadenas vacías o los espacios en blanco.

En este punto se permite la opción de utilizar un filtro, implementado como una constante, que se define en el script, que toma el valor de un número natural. Consiste en sólo aceptar palabras que, en todo el conjunto de textos, aparezcan más de un número constante de veces. Para el modelo final, se ha elegido desactivar este filtro, configurándolo para que cada palabra aparezca mínimo una vez (incluyendo así todas las palabras), ya que al ya ser un conjunto pequeño, eligiendo una constante mayor significaría hacer más pequeño aún el conjunto de entrenamiento, limitando aún más el entrenamiento.

Finalmente, se crea un conjunto hecho con las palabras aceptadas, un conjunto hecho con las palabras eliminadas y una lista con todo el texto de las canciones seguido dividido por tokens.

4.6.3. Metadatos

Después de la tokenización, no se tienen todavía datos suficientes para entrenar un modelo. Por ello, después se procede a crear otros metadatos útiles para el modelo generador. Éstos son:

- **Diccionarios que relacionan cada palabra con su identificador numérico**: Se crean dos diccionarios de este tipo. Uno que relaciona cada token con un número único que lo identifica, y otro que hace la relación inversa. Esto es así ya que el modelo sólo acepta una entrada numérica, y sólo puede producir un resultado numérico. Por tanto, antes de introducir datos en el modelo, hay que traducir cada frase de una cadena de tokens a una cadena de números. A la salida, se traduce el número resultante de vuelta a token alfanumérico, para finalizar el ciclo de la predicción.
- **Lista de secuencias**: Son las secuencias de tokens en formato de cadena de caracteres, con las que se alimentará al modelo. Constituyen la entrada del mismo. Esta lista tiene consistencia de orden con las dos siguientes.
- **Lista de palabras anteriores**: Es la lista de palabras que preceden a cada secuencia de la lista de secuencias. Se utiliza para que a la hora de generar un resultado de cara al usuario, sólo se puedan elegir como semillas secuencias que empiecen en una línea nueva en la canción original. Para ello se comprueba que la palabra o token anterior sea un salto de línea.

- **Lista de palabras posteriores**: Esta lista de palabras también es consistente con la lista de secuencias y está constituida por los tokens que van inmediatamente después de cada secuencia. Esta lista es crucial para el entrenamiento ya que contiene la salida correcta, la que debería llegar poder generar el modelo ya que es la que viene después de la secuencia en el texto original. Es a partir de esta palabra gracias a la cual el modelo la cual aprende.

Una vez generados, se guardan la lista de secuencias, las listas de palabras anteriores y posteriores, los diccionarios que relacionan palabras con sus índices y el conjunto de palabras aceptadas en el disco duro usando a la librería Pickle. Con esto ya se tiene la información suficiente para entrenar un modelo.

4.7. Entrenador de modelos

Después de generar la metainformación del corpus de texto mediante el preprocesador, el siguiente paso es entrenar un modelo. Para ello se ha implementado, a través de un script de Python, una interfaz por línea de comandos que permite gestionar varios modelos. Además, se hace uso de las funciones de *callbacks* de Keras, para modificar el transcurso del entrenamiento.

Un punto importante a tener en cuenta es que debido al tiempo que puede llevar el entrenamiento de un modelo, ha sido primordial implementar el uso de puntos de guardado para poder interrumpir y continuar el entrenamiento en cualquier momento. Otro punto importante es que para entrenar el modelo se ha usado la librería Keras sobre el back-end de Tensorflow.

4.7.1. Gestión de modelos

El script del preprocesador sólo permite un único conjunto de metadatos de manera simultánea. Para crear otro conjunto, que contenga otros artistas, hay que sobreescribir el anterior. Por contrario, esta aplicación sí que permite gestionar la creación de varios modelos coexistentes. También se puede interrumpir el entrenamiento de un modelo, conservando toda la metainformación necesaria para continuar su entrenamiento a posteriori. Además, al comenzar la ejecución del script se le muestran al usuario a qué artistas pertenecen los datos de entrenamiento.

Las siguientes subsecciones explican las dos funcionalidades principales para gestionar entrenamientos de modelos.

Nuevo entrenamiento

Al elegir la opción de nuevo entrenamiento, se le insta al usuario a que introduzca el nombre que quiere dar al modelo nuevo. Una vez elegido un nombre, se pueden dar tres casos:

- El nombre coincide con un modelo cuyo entrenamiento ya ha **completado**: En este caso se puede elegir o sobreescribir ese modelo, elegir otro nombre, o salir de la aplicación.
- El nombre coincide con un modelo cuyo entrenamiento ha sido **interrumpido**: En este caso se le brinda al usuario las opciones de elegir otro nombre, sobreescribir ese modelo, salir de la aplicación o continuar entrenándolo, con la metainformación que le corresponde.
- Es un nombre **único**: En este caso, se inicializa un nuevo entrenamiento y se procede a entrenar un modelo.

Si se sobreescribe un modelo ya entrenado o si se empieza uno completamente nuevo, se procede cargar los datos necesarios para entrenar el modelo. También se hace la división de los datos de entrenamiento en dos conjuntos. Uno de entrenamiento, que servirá para que el modelo aprenda, y otro de validación, que servirá para comprobar la precisión del modelo sobre datos con los que no ha sido entrenado.

Continuar entrenamiento

Si se elige esta opción, se le presenta al usuario la lista de modelos con posibilidad de continuar su entrenamiento. Junto a cada entrada de modelo, se indican los artistas con cuyas letras se entrenará. En este caso vale con escribir el número correspondiente al modelo que queramos elegir para continuar con su proceso de aprendizaje.

4.7.2. Callbacks

Los callbacks de Keras son objetos que implementan funciones que se pueden ejecutar durante el entrenamiento de un modelo. En este caso se usan dos.

Por una parte, está el callback *EarlyStopping*. Éste hace que si después de un cierto número de épocas, la métrica elegida no mejora, se para el entrenamiento, incluso aunque no se haya llegado a entrenar el número de épocas máximo. Este parámetro es la paciencia que tiene el modelo, y en esta implementación toma el valor de 10 épocas, ya que mostró dar buenos resultados tras varios ensayos. La métrica elegida para controlar si se para el entrenamiento es el valor de la precisión del modelo sobre el conjunto de validación.

Por otra parte, está el callback *ModelCheckpoint*. Es éste el que permite guardar puntos de control durante el entrenamiento. Está configurado de tal manera que se guarde siempre la mejor versión del modelo. En este caso, también se toma como referencia el valor de la precisión del modelo sobre el conjunto de validación. Cuando este valor alcanza un nuevo máximo, se guarda el estado del modelo en ese momento. Cabe destacar que el nombre de los archivos de guardado indica cuántas épocas de entrenamiento le quedan al modelo, información que sirve a la hora de continuar su entrenamiento en caso de interrupción.

Estos callbacks son parámetros para la función de entrenamiento.

4.7.3. Entrenamiento

Antes de entrenar el modelo, se tiene que asegurar que exista la estructura de datos necesaria para continuar su entrenamiento en caso de interrupción. Para ello, se tiene que asegurar que exista una carpeta con su nombre, dentro de otra carpeta llamada *checkpoints*. Despúes de inicializar el modelo y sus pesos, se guarda en ésta el valor de los pesos después de la inicialización, en caso de que el entrenamiento se interrumpa antes de completar la primera época. Junto a esto, se guarda toda la metainformación del corpus de texto, así como los conjuntos de entrenamiento y validación, necesaria para continuar el entrenamiento en el caso de que se aborte el entrenamiento.

El modelo se inicializa con la arquitectura mostrada en la Figura 4.1. Aquí entran en juego varios parámetros. Por una parte, se configura el modelo para que el tamaño de la entrada y de la salida sean iguales al del conjunto de palabras aceptadas. Por otra parte, se configura la capa de *Dropout*, que sirve para que no se produzca sobreajuste, para que en cada época se obvie la actualización del 20 % de los pesos.

Una vez inicializado el modelo, se comienza a entrenar éste, pasándole los dos *callbacks*, y, además, configurando el número máximo de épocas a entrenar a 100. Los datos le son suministrados por lotes, para aprovechar las capacidades de paralelización de operaciones de Tensorflow. Los lotes tienen un tamaño fijo de 32 muestras de datos, y son creados por una función generadora, que se los suministra al modelo. Otra cualidad de la función generadora es evitar que se carguen demasiados datos simultáneos en la RAM, lo cual podría llevar a problemas de memoria, sobre todo en casos como éste, ya que se trabaja con un gran volumen de datos. El estado del entrenamiento se puede seguir por pantalla ya que hay una representación gráfica que proporciona la librería de Tensorflow.

Cabe destacar que todos los parámetros han sido decididos por prueba y error, además de por recomendación en [29].

4.7.4. Postentrenamiento

Después de que haya finalizado un entrenamiento, ya sea porque se han cumplido los requisitos del callback *EarlyStopping*, o porque se haya alcanzado el máximo número de épocas, se deben realizar los pasos adecuados para indicar al gestor de modelos que ya no se puede seguir entrenando. Para ello se borrará la carpeta con su nombre de la carpeta *checkpoints*. A su vez, se creará otra carpeta, otra vez con el nombre proporcionado al modelo, que contendrá los archivos necesarios para generar texto con él. Estos archivos son:

- El modelo serializado.
- El conjunto de palabras aceptadas.
- Los diccionarios que relacionan palabras y sus identificadores.
- Las lista se secuencias, con sus correspondientes listas de palabras anteriores y posteriores.
- El nombre de los artistas con cuyas canciones se ha entrenado el modelo.

4.8. Generador de líricas

La secuencia de aplicaciones anteriores concluye produciendo los modelos que se utilizan en esta aplicación. Al ejecutar este script de Python, los modelos entrenados se le muestran al usuario para que elija con cuál quiere realizar una generación. Además, se indica con qué artista o artistas se ha entrenado cada modelo para ayudar a la elección.

Una vez seleccionado el modelo, se cargan los datos que previamente guardó la aplicación de entrenamiento. Esta vez, como se comentó antes, se utiliza la lista de palabras anteriores para averiguar qué secuencias empiezan donde comienza un verso. Esto se ha decidido hacer así para hacer que quede una generación más natural. De entre todas estas secuencias, se le muestran al usuario siete, cargadas de modo aleatorio, para que elija una de ellas como semilla. También tiene la opción de volver a cargar siete nuevas secuencias de modo aleatorio, o de coger una semilla aleatoria sin verla antes.

Habiendo seleccionado ya una semilla, se le pide al usuario introducir el número de palabras que se quiere generar. Después, se le pide introducir un número real que servirá como parámetro de diversidad de la generación.

La diversidad es un alias que se usa para la temperatura del sampleo, que se utiliza a la hora de realizar predicciones. Con los parámetros aportados por el usuario y los metadatos sobre secuencias y palabras ya disponibles, se procede a la generación de texto. Un ejemplo del resultado obtenido se puede ver en la Figura 4.2

```
++++++++++++++++++++++++++++++
Generando texto con letras de [Walls]
----- Diversidad: 0.6
----- A partir de: y si algún día me ves corriendo y no sabes

y si algún día me ves corriendo y no sabes quién soy
 sólo mírame con mis sonrisas y mis sonrisas y mis cara
 de esta siempre
 con esta cara de niñato
 con los modales modales que no tendré jamás
 mírame con mi coraza de falso prepotente
 y el cosquilleo de cuando estás enfrente
 con las historias que tengo que contar
 mírame con mis sonrisas y mis sonrisas y mis sonrisas y si le esta esta cara
 de mentira con mis días
 y si el tiempo no se acaba
 solo háblame de ti solo háblame de ti
 yeah ni yeah
```

FIGURA 4.2: Ejemplo de generación de rap.

4.9. Aplicación Web

La aplicación web cumple la misma funcionalidad que el generador de texto, pero pasando de tener una interfaz por línea de comandos a una interfaz gráfica. Cabe destacar que se reutilizó gran cantidad del código del script de generación, pero son dos aplicaciones sin interconexión alguna, más allá de su funcionamiento similar. A parte del cambio de interfaz, también se necesitó utilizar un framework para cubrir la funcionalidad de servidor de aplicaciones y evitar tener que hacerlo desde cero. Como se comentó anteriormente, para esto se utilizó el microframework llamado Flask. En combinación con el plan de alojamiento gratuito de Heroku, que permite cargar una aplicación de hasta 500 MB y le da su propio dominio, se pudo desplegar una una aplicación que demostrara el uso del generador de líricas.

La única diferencia con el script generador de modelos es que no se puede escoger modelo, ya que se quiso emular el hecho de tener un modelo terminado y ponerlo en producción. En este caso se escogió uno de los modelos, entrenados con las líricas de *Walls* y se desplegó en una aplicación web de una única página que implementa toda la funcionalidad.

La página contiene un recuadro con información, en la que se le explica al usuario el procedimiento para generar líricas. En este caso, el único requisito es elegir una de las semillas. Se pueden cargar nuevas semillas de modo aleatorio, al igual que en el script, gracias a la técnología AJAX. También es posible coger una semilla aleatoria.

A parte de esto, se ofrecen dos campos, para introducir el número de palabras a generar y la diversidad, respectivamente. Esta vez, estos campos tienen un valor por defecto que es visible en cada celda, que para la diversidad es 0.4 y para el número de palabras es 100.

Una vez que se envía el formulario para generar la letra pulsando el botón indicado. Se hace una petición asíncrona al servidor web, usando AJAX nuevamente, que responderá con información sobre la generación y la letra creada por el modelo. En la información, se volverá a indicar el artista con el que fue entrando el modelo, y el valor de los parámetros elegidos en el formulario recibido por la aplicación.

Se puede observar una captura de pantalla de la implementación final de la aplicación web en el Anexo A. Esta muestra el resultado de una generación de 100 palabras con el modelo entrenado con letras de Walls y parametrizado con una diversidad con valor 0.8.

Capítulo 5

Pruebas

En este capítulo se describirán las pruebas que se llevaron a cabo durante el desarrollo de este proyecto.

5.1. Cuaderno de pruebas

Antes de completar el web scraper se decidió conseguir una idea general del proyecto. Se consideró útil hacer un pequeño prototipo antes de implementar una aplicación grande, para ver las grandes implicaciones de diseño y para prevenir posibles errores que pudiesen ocurrir. Para ello, en este caso utilizamos la herramienta Jupyter Notebooks. Usando la extensión de Visual Code Studio que permite trabajar con cuadernos de este tipo, creamos un archivo para hacer las pruebas necesarias antes de implementar el proyecto final.

La idea era llevar a cabo una representación minimalista del *pipeline* de puesta en producción de un modelo de generación de texto. En este cuaderno, el usuario no tiene ninguna capacidad de decisión sobre el resultado final ya que está todo programado de manera estática, permitiendo una ejecución de principio a fin, sin interrupciones para solicitar input al usuario. En este apartado no se describirá en profundidad el objetivo y funcionamiento de cada parte, ya que a ese efecto están las secciones de cada aplicación del Capítulo 4.

5.1.1. Letras y preprocesamiento

Se comenzó haciendo una descarga de todas las letras del artista llamado *Dudu*, para ver como funcionaría el *Web Scraper*. Una vez comprendido el funcionamiento de éste, se implementó, para tener todas las letras en el disco duro.

Siguió el preprocesado de texto. Está parte sirvió para comprender a qué modificaciones se debía someter el texto para normalizarlo. Finalmente se acabó con 8 técnicas

de preprocesado que ya se comentaron en su propia sección. Se utilizó una canción de *Natos & Waor*, *El Jincho* y *C Tangana* respectivamente para ver cómo afectaba cada técnica de procesado al texto, haciendo las modificaciones necesarias si hacía falta.

Al final del preprocesamiento se tiene un conjunto de frases de tokens, que son palabras entendibles por el modelo. Con ellas se puede comenzar a entrenar el modelo, habiendo inicializado sus funciones de *callback*, que son objetos de Keras que añaden funcionalidad al entrenamiento, como permitir realizar puntos de guardado o una finalización temprana del entrenamiento.

5.1.2. Tiempo de ejecución del entrenamiento

Una vez comenzado el entrenamiento se hizo aparente algo que no se había tenido realmente en cuenta hasta ahora. El entrenamiento del modelo requiere muchos recursos de procesamiento y tiempo. Por ello se decidió facilitar este proceso aprovechando la capacidad de Tensorflow de ejecutarse sobre la tarjeta gráfica.

Para hacer compatible el entorno de ejecución local con la ejecución de Tensorflow sobre GPU, se siguió el tutorial oficial de Tensorflow. [34]. Para ello se tuvo que validar los siguientes requisitos:

- **Tener una tarjeta gráfica habilitada para la tecnología CUDA.** Se utilizó una tarjeta gráfica *Nvidia GTX 1660 Ti* de la que ya se disponía.
- **Instalar el controlador de Nvidia adecuado para la tarjeta gráfica.** Se instaló el controlador con versión 465.89.
- **Un kit de herramientas CUDA compatible.** Se instaló la versión 11.
- **El SDK de cuDNN**. Se instaló la versión 8.0.4

Una vez habilitado el uso de la tarjeta gráfica para entrenar el modelo, los tiempos de procesamiento se redujeron aproximadamente a la mitad. Aún así el entrenamiento del modelo completo exigía unos tiempos totalmente desorbitados. Para el entrenamiento del modelo completo se habría necesitado una ejecución ininterrumpida de treinta días, lo cual era a todos efectos inviable para el desarrollo de este proyecto. Por ello, como luego se verá, el demostrador desplegado en la aplicación web ha sido entrenado únicamente con las letras de *Walls*. Aún así, el código permite el entrenamiento de un modelo con todas las letras disponibles, pero se aconseja tener un servidor u ordenador potente.

Investigando el tema del tiempo de ejecución, se exploraron otras posibilidades, como la de utilizar un servicio de computación en la nube para el entrenamiento. Al tener un presupuesto nulo se probaron únicamente planes gratuitos. Se hicieron pruebas con el servicio de aprendizaje profundo de Google Cloud y con Amazon Sagemaker, pero en

ambos casos, el servicio de prueba gratuita resultaba más lento que la ejecución en local asi que se desechó esta posibilidad.

5.2. Pruebas durante el desarrollo de los scripts

Durante el desarrollo se siguió metódicamente el diseño planteado. Una vez implementado cada requisito, se hicieron las pruebas y modificaciones que fuesen necesarias hasta lograr el funcionamiento deseado.

5.3. Estadísticas del modelo

En esta sección se describen las características del modelo final, el modelo que se desplegó en la aplicación web. La estadística de precisión hace referencia al porcentaje de veces que el modelo predice la palabra correcta que viene después de la secuencia de entrada, según el texto original.

- Precisión: 71.72 %
- Tamaño: 29 MB
- Tiempo de entrenamiento: 20 minutos

5.4. Pruebas durante el desarrollo de la aplicación web

Un problema que se encontró en la fase de despliegue de la aplicación web fue el límite de 500 MB que tiene establecido Heroku para el tamaño de la aplicación desplegada. En los primeros intentos, no permitió subir la aplicación al servidor. El problema no residía tanto en la aplicación en sí como en el entorno de programación necesario, que ocupa la mayor parte del cupo de memoria. Finalmente se llegó a la conclusión de que el problema se podía solucionar cambiando la versión de la librería Tensorflow. Esta por defecto viene tanto con la implementación para para CPU como la implementación GPU, lo cual ocupa mucho espacio. Como en Heroku no se puede hacer uso de ninguna tarjeta gráfica, al menos en la versión gratuita, se terminó instalando solamente la versión para CPU. Una vez hecho esto se desplegó la aplicación correctamente. Se puede ver las características de la aplicación desplegada en la Figura 5.1.

La aplicación web fue simple de implementar, ya que antes de desplegarla en el servidor Heroku, se hicieron las pruebas necesarias en local. Ahí se hicieron pruebas en la interfaz, intentando hacer que fallase el programa.

FIGURA 5.1: Información relativa al despliegue de la aplicación web en Heroku.

Por ejemplo, uno de los fallos encontrados fue que se permitía meter caracteres no numéricos en las entradas correspondientes al número de palabras a generar y la diversidad. Esto provocaba un error en el servidor, pero se solucionó rápidamente con una comprobación de errores implementada en *Javascript*.

5.5. Resultados de generación

Para probar el rendimiento de los modelos entrenados, se utilizó la aplicación web, y con una misma semilla, se hicieron consultas de generación con la misma longitud de 100 palabras, pero incrementando la diversidad en una décima, partiendo de 0.1 y terminando en 1. Los resultados se pueden ver en el Anexo B. La semilla elegida corresponde a la canción *Luna de Miel* de Walls en colaboración con Robledo ([35]). Al entrenar con un corpus tan pequeño, las probabilidades de que se repitan resultados entre distintas generaciones es muy alta, ya que para una misma palabras, se han visto pocas variaciones de cual puede ser la siguiente palabra.

Por ello, en esta prueba, se ve como coinciden los tres primeros versos en todos los casos, que corresponden a las mismas frases que siguen a la semilla en la canción de la que procede. A parte, a medida que se va incrementando la diversidad, se ve como va teniendo menos sentido semántico el texto creado. Esto se debe a que es más probable, a la hora de generar cada palabra, de elegir una que se haya visto muy pocas veces seguida de la generada justo antes. Esto, junto con el hecho de haber visto pocas veces cada palabra, resulta en secuencias de frases con menos sentido.

Por otro lado, algo que el modelo ha aprendido bastante bien es la estructura de los versos. Gracias a que se ha usado un token de salto de línea, se puede distinguir que se sigue una estructura de separación en varias líneas.

Capítulo 6

Conclusiones y trabajo futuro

En este capítulo final de la memoria, se expondrán las conclusiones realizadas después de haber completado este proyecto. Además se propondrán posibles modificaciones que se puedan hacer en un futuro.

6.1. Conclusiones

En conclusión, se ha cumplido el objetivo de este TFG, que es la implementación de una RNN que genere líricas en español.

Un aspecto obvio de la implementación de un modelo de inteligencia artificial es la necesidad de modularizar el trabajo. En este caso, se ha dividido el proyecto en cinco partes, que están constituidas por los cuatro scripts y por la aplicación web. Dividir el proyecto ha servido para tener un enfoque más organizado a la hora de trabajar. Si se hubiera planteado la aplicación como un único programa, habría sido mucho más difícil tanto su diseño como su implementación y pruebas.

Otro aspecto crucial para un proyecto de inteligencia artificial es tener buenas herramientas para permitir la creación de mejores modelos de manera iterativa e intuitiva. En este caso por ejemplo, se ha desarrollado una aplicación propia para la gestión de modelos. Así se pueden concentrar los esfuerzos en la mejora de los modelos.

También es importante contar con recursos para el proyecto. Por una parte se necesitan buenos recursos económicos, para poder proporcionar potencia computacional, ya que así el proceso de iteración de modelos se vuelve mucho más ágil. Por otra, se necesitan buenos recursos en términos de datos de entrenamiento. Esto es así ya que los datos son el componente clave para que una estructura de IA como ésta pueda aprender.

Por otra parte, se debe incidir en la importancia de la investigación del estado del arte, ya que es muy valioso el hecho de trabajar con tecnología puntera, para poder conseguir

los mejores resultados.

En relación a los objetivos propuestos al inicio del proyecto, se han conseguido todos. Se ha recopilado una base de datos de canciones reales mediante un *Web Scraper* y se ha investigado el estado actual de los desarrollos más recientes en el ámbito de la generación de líricas. También se han realizado varias iteraciones de modelos usando las herramientas que se han desarrollado para fijar los parámetros de entrenamiento. Este ciclo ha acabado con un modelo que se ha desplegado en una aplicación web para demostrar su funcionamiento, además de una aplicación por línea de comandos con la que se puede probar cada modelo generado.

6.2. Trabajo futuro

En esta sección se describen posibles modificaciones futuras que servirían para mejorar o extender la funcionalidad de este proyecto. Estas modificaciones son:

- *Unir todos los scripts en un sólo programa con interfaz visual*: Esto ayudaría a hacer más accesible la aplicación a personas menos afines a la informática, ampliando la base de potenciales usuarios. Esta aplicación se podría desplegar en un servidor para que varios trabajadores o investigadores pudiesen entrenar modelos de manera telemática.
- *Implementar un sistema de cuentas y una base de datos de letras generadas en la aplicación web*: Para hacer la aplicación web más interactiva, se podría implementar un sistema de cuentas por usuarios. Cada cuenta podría guardar las letras que genere, e incluso compartirlas con otros usuarios.
- *Invertir en recursos computacionales*: Para llegar a iterar sobre varios prototipos de modelos de manera más eficaz, sería clave usar un equipo de mayor potencia computacional, como por ejemplo, un servidor con varias tarjetas gráficas en paralelo.
- *Incluir tokens de fin de canción y fin de párrafo*: Esto serviría para que el modelo aprenda a generar canciones con una estructura de principio a fin.
- Si se decide mantener una aplicación estática, como ahora, se puede *mejorar el rendimiento realizando toda la generación en el cliente*, por ejemplo, usando la librería de Tensorflow para Javascript [36]. Así se podría maximizar los tiempos de respuesta, aliviando la carga del servidor. Otra ventaja es que se podría usar servicios de hosting estático que ofrecen mayor velocidad de conexión que Heroku, como Github Pages.
- *Evaluar el rendimiento del modelo basándose en encuestas*.

Bibliografía

[1] Tanya Lewis. *A Brief History of Artificial Intelligence*. Dic. de 2014. URL: https://www.livescience.com/49007-history-of-artificial-intelligence.html (visitado 10-05-2021).

[2] D. Pawade, Avani M. Sakhapara, Mansi Jain, Neha Jain y Krushi Gada. «Story Scrambler - Automatic Text Generation Using Word Level RNN-LSTM». En: *International Journal of Information Technology and Computer Science* 10 (2018), págs. 44-53.

[3] Mao Li, Jiancheng Lv, Jian Wang y Yongsheng Sang. «An Abstract Painting Generation Method Based on Deep Generative Model». En: *Neural Processing Letters* 52.2 (jun. de 2019), págs. 949-960. DOI: 10.1007/s11063-019-10063-3. URL: https://link.springer.com/article/10.1007%2Fs11063-019-10063-3.

[4] Lucas Whittaker, Kate Letheren y Rory Mulcahy. «The Rise of Deepfakes: A Conceptual Framework and Research Agenda for Marketing». En: *Australasian Marketing Journal* (mar. de 2021), pág. 183933492199947. DOI: 10.1177/1839334921999479. URL: https://journals.sagepub.com/doi/abs/10.1177/1839334921999479.

[5] Madeleine Hernandez, Sylvain Kahane, Serge Fleury y Kim Gerdes. *Cover Page Automatic Generation of Hip-Hop and Rap Lyrics A backwards n-grams model using its own corpus to produce original rap lyrics*. 2017. URL: http://www.tal.univ-paris3.fr/plurital/memoires/HERNANDEZ_madeleine-RD-1718.pdf.

[6] *The Original Hip-Hop Lyrics Archive*. 2021. URL: http://ohhla.com/ (visitado 10-05-2021).

[7] K. Watanabe y M. Goto. «Lyrics Information Processing: Analysis, Generation, and Applications». En: *NLP4MUSA*. 2020.

[8] Eric Malmi, Pyry Takala, Hannu Toivonen, Tapani Raiko y Aristides Gionis. «DopeLearning». En: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (ago. de 2016). DOI: 10.1145/2939672.2939679. URL: https://arxiv.org/abs/1505.04771.

[9] Peter Potash, Alexey Romanov y Anna Rumshisky. «GhostWriter: Using an LSTM for Automatic Rap Lyric Generation». En: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Lisbon, Portugal: Association for Computational Linguistics, sep. de 2015, págs. 1919-1924. DOI: 10.18653/v1/D15-1221. URL: https://www.aclweb.org/anthology/D15-1221.

[10] Ralf C. Staudemeyer y Eric Rothstein Morris. «Understanding LSTM - a tutorial into Long Short-Term Memory Recurrent Neural Networks». En: *CoRR* abs/1909.09586 (2019). arXiv: 1909.09586. URL: http://arxiv.org/abs/1909.09586.

[11] *Genius API*. 2014. URL: https://docs.genius.com/ (visitado 10-05-2021).

[12] DocsPreview. *Estado de la versión de Windows*. 2021. URL: https://docs.microsoft.com/es-es/windows/release-health/ (visitado 10-05-2021).

[13] *Documentation for Visual Studio Code*. Abr. de 2016. URL: https://code.visualstudio.com/docs (visitado 10-05-2021).

[14] *Documentation Heroku Dev Center*. 2021. URL: https://devcenter.heroku.com/categories/reference (visitado 10-05-2021).

[15] *Git*. 2021. URL: https://git-scm.com/ (visitado 10-05-2021).

[16] *Begginners Guide Python*. 2019. URL: https://wiki.python.org/moin/BeginnersGuide/Overview (visitado 10-05-2021).

[17] *Project Jupyter | About Us*. 2014. URL: https://jupyter.org/about (visitado 10-05-2021).

[18] *Flask Documentation*. 2021. URL: https://flask.palletsprojects.com/en/2.0.x/ (visitado 10-05-2021).

[19] *Postman Learning Center*. 2021. URL: https://learning.postman.com/docs/getting-started/introduction/ (visitado 10-05-2021).

[20] *pandas documentation*. 2021. URL: https://pandas.pydata.org/docs/ (visitado 10-05-2021).

[21] *Request documentation*. 2013. URL: https://docs.python-requests.org/es/latest/ (visitado 10-05-2021).

[22] *Beautiful Soup 4*. Oct. de 2020. URL: https://pypi.org/project/beautifulsoup4/ (visitado 10-05-2021).

[23] *Numpy Documentation*. 2021. URL: https://numpy.org/doc/stable/ (visitado 10-05-2021).

[24] *Regular Expressions operations - Documentation*. 2021. URL: https://docs.python.org/3/library/re.html (visitado 10-05-2021).

[25] *Pickle Documentation*. 2021. URL: https://docs.python.org/3/library/pickle.html (visitado 10-05-2021).

[26] *Glob Documentation*. 2021. URL: https://docs.python.org/3/library/glob.html (visitado 10-05-2021).

[27] Keras Team. *Keras documentation: Keras API reference*. 2021. URL: https://keras.io/api/ (visitado 10-05-2021).

[28] *lyricsgenius - Pypi*. Abr. de 2021. URL: https://pypi.org/project/lyricsgenius/ (visitado 10-05-2021).

[29] enriqueav. *enriqueav/lstm_lyrics*. Ene. de 2019. URL: https://github.com/enriqueav/lstm_lyrics/blob/master/lstm_train_embedding.py (visitado 10-05-2021).

[30] Piotr Bojanowski, Armand Joulin y Tomás Mikolov. «Alternative structures for character-level RNNs». En: *CoRR* abs/1511.06303 (2015). arXiv: 1511.06303. URL: http://arxiv.org/abs/1511.06303.

[31] *PEP 574*. 2018. URL: https://www.python.org/dev/peps/pep-0574/#abstract (visitado 10-05-2021).

[32] *The HDF5 Library and File Format*. 2018. URL: https://www.hdfgroup.org/solutions/hdf5/ (visitado 10-05-2021).

[33] Massimo Caccia, Lucas Caccia, William Fedus, Hugo Larochelle, Joelle Pineau y Laurent Charlin. «Language GANs Falling Short». En: *CoRR* abs/1811.02549 (2018). arXiv: 1811.02549. URL: http://arxiv.org/abs/1811.02549.

[34] *Compatibilidad con GPU - Tensorflow*. 2021. URL: https://www.tensorflow.org/install/gpu?hl=es-419 (visitado 10-05-2021).

[35] Walls. *Walls Ft. Robledo - LUNA DE MIEL (Videoclip Oficial)*. Abr. de 2019. URL: https://www.youtube.com/watch?v=PDterKwfWMw (visitado 10-05-2021).

[36] *Tensorflow.js*. 2020. URL: https://www.tensorflow.org/js?hl=es-419 (visitado 10-05-2021).

[37] Keras Team. *Keras documentation: Dropout layer*. 2021. URL: https://keras.io/api/layers/regularization_layers/dropout/ (visitado 10-05-2021).

Anexos

Anexo A

Interfaz Web App

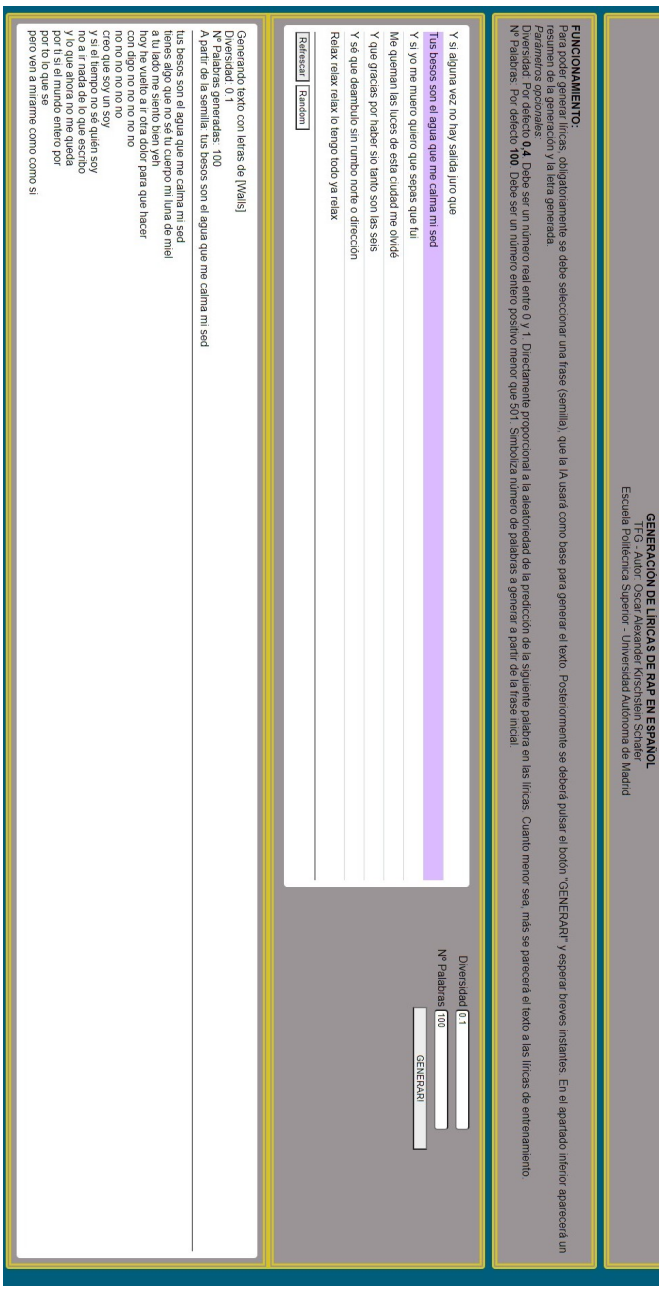

FIGURA A.1: Captura de pantalla de la aplicación web.

Anexo B

Resultados de generación

En este Anexo se muestran una serie de resultados de generación. El modelo empleado es el modelo entrenado únicamente con letras de Walls. La frase semilla utilizada es: "Tus besos son el agua que calma mi sed". Se generan 100 palabras cada vez, y se prueba a utilizar distinta diversidad, partiendo de 0.1 y llegando, mediante incrementos de una décima, al 1.

- **Diversidad = 0.1**
  tus besos son el agua que me calma mi sed
  tienes algo que no sé tu cuerpo mi luna de miel
  a tu lado me siento bien yeh
  hoy he vuelto a ir otra dolor para que hacer
  con digo no no no no no
  no no no no no no
  creo que soy un soy
  y si el tiempo no sé quién soy
  no a ir nada de lo que escribo
  y lo que ahora no me queda
  por ti si el mundo entero por
  por to lo que se
  pero ven a mirarme como como si

- **Diversidad = 0.2**
  tus besos son el agua que me calma mi sed
  tienes algo que no sé tu cuerpo mi luna de miel
  a tu lado me siento bien yeh
  hoy he vuelto a ir otra dolor para que hacer
  con digo no no no no no
  no no no no no no no
  creo que soy un soy
  soy de tu lado me siento bien te voy a comprar la casita en la playa
  con bares y vistas hacia el mar no no no no no no no

yeah
no no no no no
no no no no no no

- **Diversidad = 0.3**
  tus besos son el agua que me calma mi sed
  tienes algo que no sé tu cuerpo mi luna de miel
  a tu lado me siento bien yeh
  hoy he vuelto a ir otra para casa que casa de que no
  no quiero que quiero
  que se rieron ahora el orgullo ha opacado el amor que se transparentaba
  quiero enloquece en tu mirada y ahogarme en la lava que se esconde por tu carmín
  de si el mundo entero entero
  por un zapato de cartón
  por un zapato de cartón
  a si me muero
  quiero que sepas que fui

- **Diversidad = 0.4**
  tus besos son el agua que me calma mi sed
  tienes algo que no sé tu cuerpo mi luna de miel
  a tu lado me siento bien yeh
  iguala ven a mirarme como sabes a erizarme la piel
  tus besos son el agua que me calma mi sed
  tienes algo que no sé tu cuerpo mi luna de miel
  a tu lado me siento bien yeh
  hoy he vuelto a mirarme otra otra excusa para para verte
  digo no no no no no no
  voy a dejar que escapen mediante aquello que inventan
  y digo no no no no no

- **Diversidad = 0.5**
  tus besos son el agua que me calma mi sed
  tienes algo que no sé tu cuerpo mi luna de miel
  a tu lado me siento bien yeh
  hoy he vuelto a vuelto a perder en la ruta 66 de tu rímel a tu mirá
  siendo tus ojos las únicas piedras con las que tropezar
  llorando agua dulce acostumbrado a nadar de alta alta mar
  te juro que te juro más te juro
  y más que algún alma y cuerpo cuerpo de miel
  a tu tu lado me siento
  y tú al bostezo y me ves porque
  y si estoy

- **Diversidad = 0.6**
  tus besos son el agua que me calma mi sed

tienes algo que no sé tu cuerpo mi luna de miel
a tu lado me siento bien yeh
iguala ven a mirarme como sabes a erizarme la piel
tus besos son el agua que me calma mi sed
tienes algo que no sé tu cuerpo mi luna de miel
a tu lado me siento bien yeh
hoy he vuelto a ir otra vuelto a casa que su
no quiero que no
no me diga de voy a a ir para mezclar
y digo no no no no no no

- **Diversidad = 0.7**
  tus besos son el agua que me calma mi sed
  tienes algo que no sé tu cuerpo mi luna de miel
  a tu lado me siento bien yeh
  iguala otra vez otra vez en el primero
  que solo te ha ha pasado y el centro con amor esto se
  quiero que preguntan lo que ha pedir
  lo que nos sé la peor de las
  y de decirte que lo que menos la momento
  porque yo por haber vuelto y tan sólo entiende
  que soy una razón
  pa que no nos llame
  así que dime qué va qué va qué

- **Diversidad = 0.8**
  tus besos son el agua que me calma mi sed
  tienes algo que no sé tu cuerpo mi luna de miel
  a tu lado me siento bien yeh te voy a comprar la casita en la playa
  con bares y vistas hacia el mar no no no no no no no no
  no no no no
  creo que soy un resto de los par
  con mi de mis pende de un tanto
  de vuelta al bañado en la exacto
  y si hacia el que escribo
  que el ha escuchado antes de alzar la alzar
  y si pierdo la pierdo

- **Diversidad = 0.9**
  tus besos son el agua que me calma mi sed
  tienes algo que no sé tu cuerpo mi luna de miel
  a tu lado me siento bien yeh
  hoy he vuelto a ir bien seguir y al azar
  a todo esto lo malo que pero no
  no necesito a ir lo que no quiero

pero y si estoy sin ti
hasta que el centro de la mentira y al azar
y alma al lugar de lo alma
pero qué va y te retroceder
y le parar parar parar
niña en que nos con esto entero
lo siento

- **Diversidad = 1**
  tus besos son el agua que me calma mi sed
  tienes algo que no sé tu cuerpo mi luna de miel
  a tu lado me siento bien yeh
  hoy ahora me he vuelto a pensar quién soy de esto
  no dices
  dame a razón
  pa una marcharme corriendo a evitar tu jugada
  dame una razón
  solo quiero seguir a tu lado y si el tiempo se acaba
  solo háblame de ti solo háblame de ti
  solo háblame de ti solo háblame de ti
  yeah eh yeah
  solo háblame de ti no no no
  yeah
  yeah

Anexo C

Manual de uso

Este anexo contiene el manual de uso para cada aplicación del proyecto, junto con capturas de pantalla para ayudar a la comprensión. Los requisitos para poder ejecutar cualquiera de los scripts, son tener instalados:

- Python 3.8.4 de 64 bits.
- Los módulos de Python indicados en el archivo *requirements.txt*.

Es destacable mencionar, que en todos los scripts de línea de comandos, se puede finalizar la ejecución en cualquier momento, siempre que se nos solicita entrada de datos. Esto se hace escribiendo la cadena *"ESC"*.

C.1. Web Scraper

Esta aplicación es la más simple ya que para utilizarla, sólo se tiene que ejecutar el script llamado *lyrics_scraper.py*. Sin embargo, hay un requisito importante, que es tener el archivo CSV que relaciona artistas e IDs. El proyecto ya viene con un archivo por defecto con 21 artistas seleccionados a mano.

Las posibles acciones son:

- **Eliminar artistas**: Se tiene que borrar la línea correspondiente del archivo CSV. Después, se siguen las instrucciones para descargar las canciones.
- **Añadir artistas**: Se tiene que conseguir el *id* que tiene en la base de datos de Genius. La guía de como conseguir el id del artista está en el Capítulo 4.2. Una vez obtenido, se insertan el nombre del artista y su *id* en la columna indicada. Después, se siguen las instrucciones para descargar las canciones.

- **Descargar canciones**: Para hacer una nueva descarga de canciones, primero hay que asegurar que no existe el archivo *canciones.pkl* dentro de la carpeta *data* dentro del directorio de trabajo del proyecto. Además, hay que eliminar el contenido de la carpeta *data/letras_raw*, si es que lo hubiese. En el caso de descargar las canciones por primera vez, no habría que hacer ninguna de estas comprobaciones. Por último, se ejecuta el script *lyrics_scraper.py*.

El resultado de la ejecución del programa es la creación del archivo *data/canciones.pkl* que contiene los metadatos auxiliares de las canciones, y los archivos con las letras de las canciones, que están en *data/letras_raw*. Éstos últimos están divididos en archivos con el nombre del artista al que pertenecen las canciones, respectivamente. Aquí, primero se nos indica de que artistas se compone el conjunto de datos procesados existente actualmente.

C.2. Preprocesador de texto

Este programa requiere que existan archivos con letras de canciones, como los descritos en la sección anterior. El primer paso para hacer un preprocesamiento de texto consiste en ejecutar el script *text_preprocessing.py*. Se nos mostrará un menú como el de la Figura C.1.

```
Ya hay un dataset con datos de las canciones de [Walls]
Elige el artista con el que quieres hacer los datos de entrenamiento del modelo.
Se elige escribiendo el número adecuado. Se puede elegir varios artistas escribiendo los números correspondientes separados por comas.
Para elegir todos los artistas, escribe TODOS. Para salir escribe ESC. Estos son los artistas disponibles:
0.- C Tangana
1.- Cecilio G
2.- Chaman
3.- Charlie
4.- Delaossa
5.- Denom
6.- Dudu
7.- Easy-S
8.- El Jincho
9.- Hard GZ
10.- Israel B
11.- Ivan Cano
12.- Kase O
13.- Lopes
14.- Natos y Waor
15.- Nikone
16.- Oktoba
17.- Recycled J
18.- Tote King
19.- Walls
20.- Yung Beef
>>
```

FIGURA C.1: Captura de pantalla del menú de la aplicación de preprocesamiento de texto.

También se nos insta a seleccionar los artistas de cuyas canciones queremos construir el conjunto de datos de entrenamiento. Como se puede ver en la Figura C.1, se enumeran los artistas de cuyas letras se dispone. Los ejemplos que se dan a continuación se basan en esta figura. Se pueden tomar dos acciones distintas:

1. **Seleccionar un artista**: Para ello, simplemente se introduce el número que le corresponde en la lista. p.e.:
   - Si se quiere elegir a Cecilio G, se introduciría: **1**.
2. **Seleccionar varios artistas**: Para ello, se introducen los números que les corresponden a los artistas deseados, separados por una coma. p.e.:
   - Si se quiere elegir a Cecilio G, Chaman y Denom se introduciría: **1, 2, 5**.

Una vez hecha una de las dos elecciones, se muestran datos relevantes al procesamiento, como en la Figura C.2. Después, el programa termina.

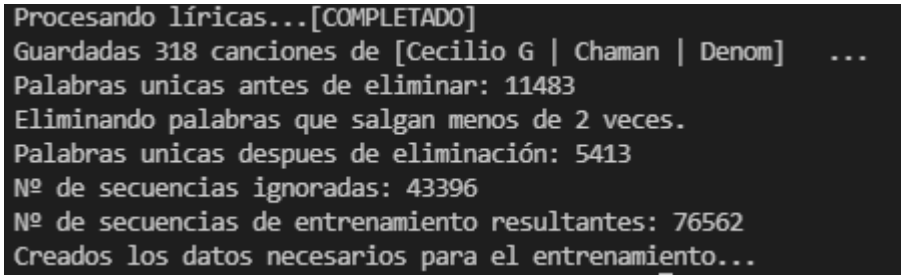

FIGURA C.2: Captura de pantalla de la información mostrada sobre el preprocesamiento de texto.

El resultado de la ejecución es la creación de dos archivos con los datos necesarios para el entrenamiento de un modelo y la generación de texto consecuentes. Estos archivos son *data/info_idx.pkl* y *data/pre_train.pkl*.

C.3. Entrenador de modelos

Esta aplicación requiere de la existencia de los dos archivos que contienen los datos de las letras procesadas, los cuales se mencionan al final de la sección anterior. Para iniciar la aplicación se ejecuta el script *lstm_training.py*, tras lo cual se nos muestra el menú de la Figura C.3.

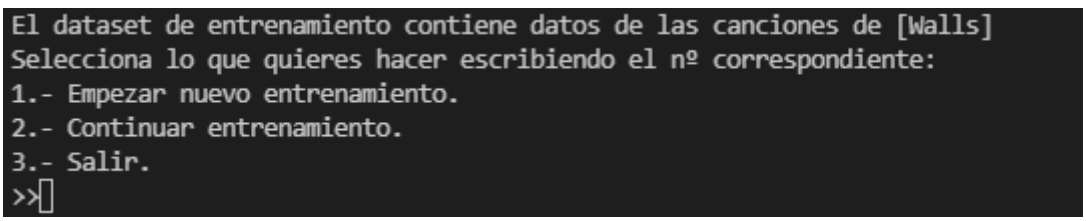

FIGURA C.3: Captura de pantalla del menú inicial de la aplicación de entrenamiento de modelos.

A parte de la posibilidad de salir del programa, se nos presentan dos opciones, que se seleccionan introduciendo el número correspondiente de la enumeración. Estas opciones se describen a continuación en sus correspondientes secciones. También se nos informa de los artistas con cuyas letras se entrenará en caso de empezar un nuevo entrenamiento.

Cabe destacar, que, una vez comenzado un entrenamiento, éste se puede interrumpir en cualquier momento pulsando la combinación de teclas **CTRL + C**, quedando guardada la mejor versión del modelo hasta ese momento. Además, existen una serie de parámetros, dentro del código del script, que se pueden modificar para alterar el entrenamiento o el modelo. Éstos se explican en detalle en el Anexo D

C.3.1. Nuevo entrenamiento

Se selecciona introduciendo un uno en el menú inicial. Tras hacerlo, se nos insta a introducir un nombre para el nuevo modelo, como se indica en la Figura C.4.

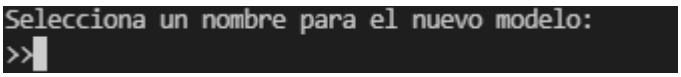

FIGURA C.4: Captura de pantalla del menú de selección de nombre del modelo nuevo.

Una vez introducido el nombre, hay tres casos:

1. **El nombre es único**: En este caso, se procede a entrenar el modelo, mostrando el proceso de entrenamiento tal y como indica la Figura C.5.
2. **El nombre coincide con el de un modelo ya entrenado**: En este caso, se nos presentan tres opciones, como se indica en la Figura C.6:
   a) **Elegir otro nombre:** Se vuelve al proceso de elegir un nombre para el modelo nuevo.
   b) **Sobreescribir modelo antiguo:** En este caso se eliminan todos los datos del modelo con el que coincide el nombre y se empieza a entrenar el modelo, con el conjunto de datos de entrenamiento actuales.
   c) **Salir:** Terminar la ejecución del programa sin entrenar un modelo.
3. **El nombre coincide con el de un modelo cuyo entrenamiento fue interrumpido**: En este caso se nos presenta un menú con cuatro opciones, como se indica en la Figura C.7:
   a) **Elegir otro nombre**: Se vuelve al proceso de elegir un nombre para el modelo nuevo.
   b) **Empezar entrenamiento de 0**: Se borran los datos de los puntos de guardado del modelo con cuyo nombre coincide, y se empieza a entrenar un modelo nuevo, con el conjunto de datos de entrenamiento actuales.
   c) **Continuar entrenando**: Se continua con el entrenamiento del modelo cuyo entrenamiento había sido interrumpido, con el conjunto de datos con el que en su momento se empezó a entrenar.
   d) **Salir**: Terminar la ejecución del programa sin entrenar un modelo.

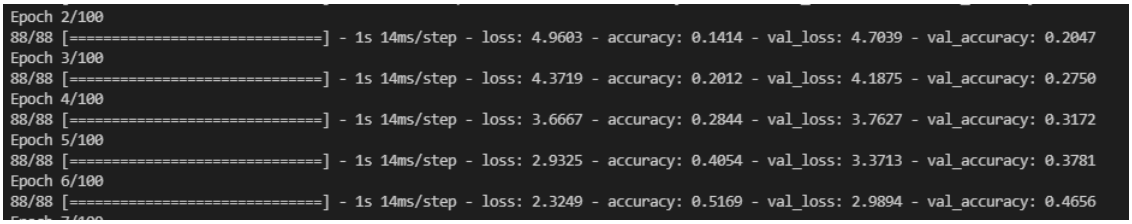

FIGURA C.5: Captura de pantalla de la información mostrada durante el entrenamiento de un modelo.

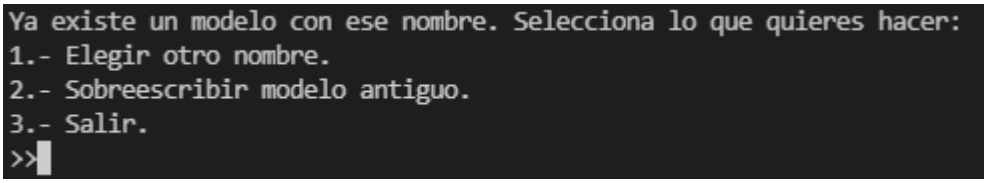

FIGURA C.6: Captura de pantalla con el menú que se muestra si el nombre elegido para el modelo nuevo coincide con el de un modelo ya entrenado.

C.3.2. Continuar entrenamiento

Se selecciona introduciendo un dos en el menú principal. Tras esto, se presenta un menú como el de la Figura C.8 presentando todos los modelos disponibles para continuar entrenando.

Para elegir uno, se debe introducir el número correspondiente de la enumeración. Una vez seleccionado, se prosigue con el entrenamiento a partir del punto en el que se interrumpió.

C.4. Generador de líricas

Este programa requiere que exista algún modelo entrenado por completo y se inicia ejecutando el script *lyrics_sampling.py*.

Una vez ejecutado, se nos presenta un menú en la terminal, que nos pide seleccionar un modelo con el que generar la lírica. Al lado de cada nombre de modelo, aparecen entre corchetes los artistas con los que se ha entrenado el modelo. Este menú se puede ver en la Figura C.9

Una vez seleccionado un modelo introduciendo el número correspondiente, se carga una selección de 6 semillas, como se puede observar en la Figura C.10 Aquí se pueden realizar tres acciones:

1. Elegir una de las seis semillas.
2. Se elige una semilla aleatoria, introduciendo el número 7 por la línea de comandos. Esta semilla es una frase aleatoria sacada del cuerpo de texto de entrenamiento.

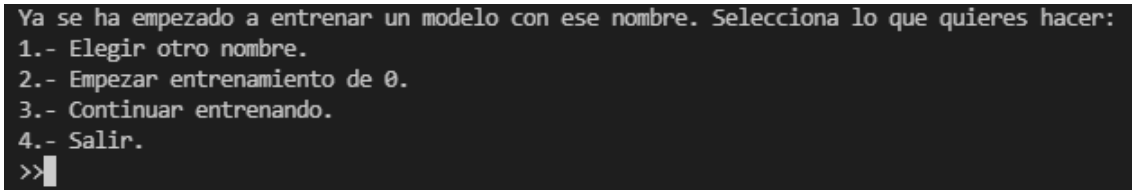

FIGURA C.7: Captura de pantalla con el menú que se muestra si el nombre elegido para el modelo nuevo coincide con el de un modelo cuyo entrenamiento ha sido interrumpido.

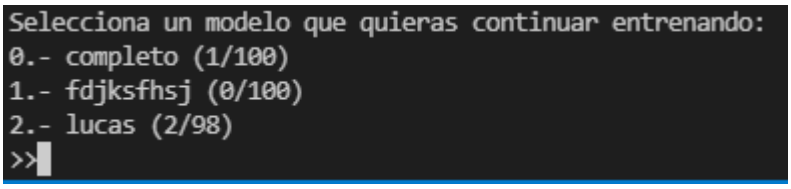

FIGURA C.8: Captura de pantalla del menú de selección de modelo que se quiere continuar entrenando.

3. Se elige recargar las semillas. En este caso, se cargan otras seis semillas de entre todas las disponibles, y se vuelven a disponer de estas tres opciones.

Una vez elegida una semilla, ésta se le muestra al usuario y se procede al siguiente paso. Este consiste en elegir primero, la cantidad de palabras a generar (que debe ser un número entre 20 y 500), y luego la diversidad con la que hacer la generación (que debe ser un número mayor que 0 y menor o igual que 1). Esto se puede observar en la Figura C.11.

Una vez realizados estos pasos, se genera el texto con los parámetros seleccionados y se nos lo muestra. Después, termina la ejecución.

C.5. Aplicación Web

Para usar el generador de líricas desplegado como aplicación web, se debe acceder a la siguiente URL:

- https://frozen-inlet-60325.herokuapp.com/

Una vez accedido, se nos muestra la interfaz, tal y como aparece en la Figura A.1. La única diferencia es que la sección blanca inferior nos aparecerá vacía, ya que no se ha generado ningún texto todavía.

Para realizar una generación, es obligatorio seleccionar una de las semillas, ya sea una de las que se muestran, o una aleatoria pulsando el botón "*RANDOM*". Además, se ofrece la posibilidad de recargar la selección de frases mediante el botón "*REFRESCAR*".

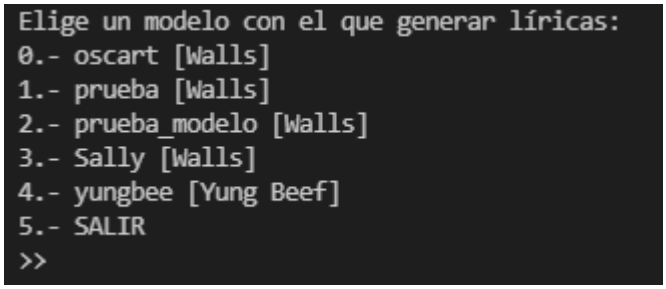

FIGURA C.9: Captura de pantalla del menú de selección de modelo con el que se quiere generar líricas.

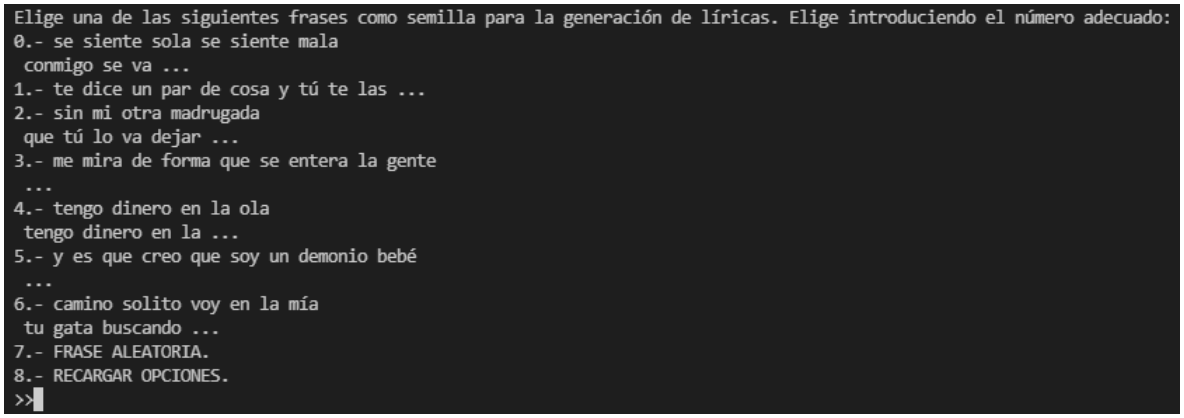

FIGURA C.10: Captura de pantalla del menú de selección de la semilla con la que se quiere generar las líricas.

Se puede continuar parametrizando la generación, introduciendo valores para la diversidad y el número de palabras a generar. Para la diversidad debe ser un número real en el rango (0,1], y para el número de palabras debe ser un número natural menor que 501. Estos dos parámetros son opcionales. En caso de no rellenarlos toman el valor de 0.4 y 100 por defecto, respectivamente.

Una vez ajustadas las variables, para realizar la generación se debe pulsar el botón "*GENERAR*!", tras lo cual se mostrará el texto generado junto a información sobre la generación.

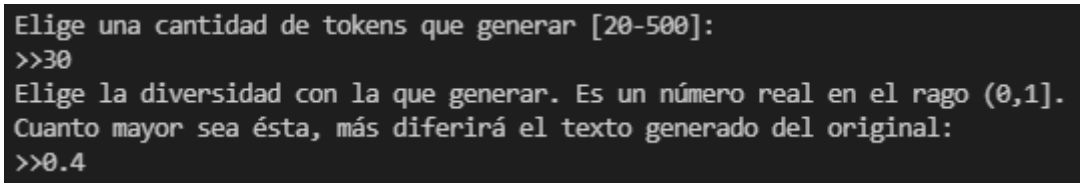

FIGURA C.11: Captura de pantalla de las entradas de los valores del número de palabras y al diversidad con las que realizar la generación.

Anexo D

Parámetros de entrenamiento

En el script de *lstm_training.py*, el código contiene ciertos parámetros que se pueden modificar para cambiar el comportamiento del modelo y del entrenamiento. En esta sección se describirán estas variables.

- **DROPOUT**: Debe tener un valor en el rango (0,1). El valor representa el porcentaje de nodos de la capa de entrada a anular durante el entrenamiento del modelo. Esto se implementa a través de la capa del modelo llamada *Dropout*, proporcionada por Keras [37]. Lo que provoca es que se consigue que el modelo aprenda a generalizar mejor.
- **NUM_EPOCHS**: Representa el número de épocas que debe entrenar como máximo el modelo.
- **TAM_BATCH**: El modelo Keras, con el fin de aprovechar la arquitectura de los procesadores modernos, permite que se procesen los datos de aprendizaje por lotes. Ésta constante simboliza el número de muestras individuales de las que se compone cada uno de los lotes.

Anexo E

Definiciones

- **Corpus**: Conjunto cerrado de textos o de datos destinado a la investigación científica.
- **Pipeline**: Una cadena de procesos conectados de forma tal que la salida de cada elemento de la cadena es la entrada del próximo.
- **Web Scraper**: Un programa que sirve para extraer información de páginas web de forma automatizada.
- **Token**: Una unidad de significado semántico/estructural.

Anexo F

Acrónimos

- **RNN** — Recurrent Neural Network
- **TFG** — Trabajo de Fin de Grado
- **API** — Aplication Programming Interface
- **IA** — Inteligencia Artificial
- **ANN** — Artificial Neural Network
- **HTTP** — HyperText Transfer Protocol
- **IR** — Information Retrieval
- **LSTM** — Long Short-Term Memory
- **RankSVM** — Ranking Support Vector Machine
- **DFD** — Diagrama de Flujo de Datos
- **Bi-LSTM** — LSTM Bidireccional
- **SO** — Sistema Operativo
- **CSV** — Comma Separated Values
- **URL** — Uniform Resource Locator
- **PaaS** — Platform as a Service